OpenZeppelin | security

Origin Native Staking Audit

ORIGIN

June 11, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-05-13 To 2024-05-27
Languages	Solidity

Total Issues	12 (7 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (1 resolved)
Low Severity Issues	1 (0 resolved)
Notes & Additional Information	7 (4 resolved)

Scope

We audited the OriginProtocol/origin-dollar repository at commit 21d5569.

In scope were the following files:

```
contracts/contracts
├── interfaces
│   ├── IDepositContract.sol
│   └── ISSVNetwork.sol
├── proxies
│   └── Proxies.sol
├── strategies
│   └── NativeStaking
│       ├── FeeAccumulator.sol
│       ├── NativeStakingSSVStrategy.sol
│       ├── ValidatorAccountant.sol
│       └── ValidatorRegistrator.sol
└── utils
    └── InitializableAbstractStrategy.sol
```

System Overview

The `NativeStakingSSVStrategy` contract introduces a new strategy backing Origin Ether (OETH) based on collecting yield from staking ETH. Strategies act as yield-bearing collateral for OETH and accumulate yield which is then distributed to OETH holders via rebasing. Traditionally, staking balances and rewards are tracked using an oracle. However, the initial native staking strategy does not include a staking oracle. To correctly infer the amount earned by staking rewards, the amount of ETH held in the contract is used to extrapolate whether the source of ETH is from a fully withdrawn validator, staking rewards, or a slashed validator. For this, the strategy classifies the amount of ETH in intervals to infer if a validator was slashed or earned rewards. These intervals have a safety buffer called the fuse. An ETH amount which is between the fuse interval indicates that a mass slashing event occurred and this would trigger a safety fuse that will pause the contracts until the accounting is manually fixed.

Due to potential inaccuracies in the accounting system, off-chain monitoring is also employed to check for unusual behavior. The system requires manual intervention by the strategist in case multiple validators are slashed in a short time frame. The accounting can be manually fixed when the contract is paused by changing the active deposited validators and consensus rewards. The protocol integrates with the SSV Network to manage validators and staking operations. The `NativeStakingSSVStrategy` contract holds SSV tokens used to pay SSV operators.

Consensus rewards are allocated to the harvester, while deposits from active validators are directed to the vault. ETH accumulates at a specific address and is converted to WETH during the `doAccounting` and `collectRewardTokens` processes. These rewards are then transferred to the harvester and then to the dripper which gradually streams the rewards to the vault.

The `NativeStakingSSVStrategy` contract shares functionalities with other OETH strategies, with functions such as `_collectRewardTokens` to gather both consensus and execution rewards and `checkBalance` to view the amount of ETH backing the strategy. Execution rewards are collected in the Fee Accumulator, which includes transaction fees and MEV rewards, whereas consensus rewards are sourced from the Beacon Chain.

Security Model and Privileged Roles

The following trust assumptions were made about the codebase during the audit:

- The registrator is trusted to select the correct validators.
- SSV network validators are trusted to behave honestly and validate Ethereum blocks correctly.
- The strategist is expected to honestly update the accounting. The manual fix can set the consensus rewards and active validators to whatever they wish.
- The strategist is relied upon to actively update accounting. Without the strategist actively responding to unexpected incidences, the contract could have inaccurate accounting and thus may direct incorrect funds to the vault and harvester, or become paused for an extended period.
- The strategist must unpause the contracts.
- It is not always possible to withdraw all the ETH from the contracts due to a staking delay.
- It is possible for the strategy to lose money due to slashing exceeding rewards. This is relevant as OETH strategies should always have a 1-1 backing.
- It is not possible to differentiate between ETH that is sent via `selfdestruct` and ETH that is sourced from staking redemptions and rewards, which means the accounting can be incorrect. The staking contracts gain ETH whenever this happens and the audit did not uncover a strong profit incentive for such an attack. However, this is still a possibility and it is difficult to fix without implementing a staking oracle.
- SSV Tokens are fully approved to the SSV contracts and these approvals cannot be revoked.

Medium Severity

M-01 Accounting Can Be Manipulated Through ETH Donation Via Self-Destruct

The `_doAccounting` function of the `ValidatorAccountant` contract extrapolates whether changes to the ETH balance of the contract come from slashing or accumulation of the staking rewards based the amount of ETH modulo 32 `ether`. Therefore, it is crucial to prevent manipulation of the ETH balance by transferring Ether.

Origin does this by only accepting ETH when the sender is the fee accumulator or the WETH token address in the `receive` function. However, it is possible to bypass these checks by using a contract that calls `selfdestruct` in the constructor with the `NativeStakingSSVStrategy` contract as the target. An attacker could use this for various impacts such as:

- Triggering the fuse even when no slashing has occurred
- Making the fuse not trigger when it is supposed to
- Making consensus rewards be registered as a slashing event

These can be fixed manually and active monitoring could mitigate the impact of such events. However, they are still a security concern and will cause the `NativeStakingSSVStrategy` contract to deviate from expected behavior.

As a long-term solution, consider using an oracle that monitors slashing and consensus rewards.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *In each of the scenarios below, the strategy profits from the attacker donating ETH to the contract. Let's look at how the Strategist would correct the accounting in each of the attack scenarios.*
>
> 1. ***Triggering the fuse even when no slashing has occurred****: In this scenario, extra ETH is donated to trigger the fuse. This can be fixed with* `manuallyFixAccounting` *that can allocate the donated ETH to consensus rewards and reset the fuse.*

> 2. ***Making the fuse not trigger when it is supposed to****: This scenario can occur in two ways:*
>
>   - *There are extra consensus rewards so the fuse should be blown. However, the donation adds extra ETH so the fuse is not blown.* `doAccounting` *incorrectly thinks the ETH is from a withdrawal from a slashed or full validator depending on how much ETH was donated. To correct this, the Strategist would pause the contract and call* `manuallyFixAccounting` *to adjust the consensus rewards and active validators upward.*
>
>   - *There are withdrawals from some validators with bigger slashed amounts than expected. This should trigger the fuse. However, the donation of the extra ETH does not trigger the fuse as it accounts for it as validator withdrawals. There is no action to be taken by the strategist here. The extra ETH from the donation is just allocated as staked ETH from a validator.*
>
> 3. ***Making consensus rewards be registered as a slashing event****: For this scenario, enough extra ETH has to be donated so* `doAccounting` *thinks that there was an extra withdrawal from a slashed validator. That is, the remaining ETH is above the fuse interval's end amount. To correct this, the Strategist has to pause the strategy and then call* `manuallyFixAccounting` *to increase the consensus rewards, increase the active validators, and reset the fuse.*

M-02 All Addresses Are Registered Validators by Default

In Solidity, an enum variable is automatically set to its first enumerated value if it is initialized without specifying a particular value. In `ValidatorRegistrator.sol`, REGISTERED is the default value of the `VALIDATOR_STATE` enum, meaning that it is possible to stake with any address even if `registerSsvValidator` were not called. Since the address would not be registered as a validator in the SSV Network, it would not perform validations correctly which could result in funds being lost due to slashing.

Consider adding `NON_REGISTERED` as the default value in the `VALIDATOR_STATE` enum.

Update: *Resolved in pull request #2081.*

Low Severity

L-01 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

Throughout the codebase, the floating pragma `solidity ^0.8.0` is used.

Consider using fixed pragma directives.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *Acknowledged but will not change. The origin-dollar repo has used Solidity ^0.8.0 for a number of years now. The same compiler has also been used which is 0.8.7. This is now quite old. We have looked at upgrading the Solidity version in the past as it does give some gas improvements. Functionally, there is not a big driver to upgrade the Solidity version so we have left it.*

Notes & Additional Information

N-01 Inconsistent Order Within Contracts

Throughout the codebase, there are multiple contracts that deviate from the Solidity Style Guide due to having inconsistent ordering of functions and layout.

Here are some examples of such contracts:

- The `InitializableAbstractStrategy` contract in `InitializableAbstractStrategy.sol` has an incorrect order of functions, modifiers, and structs.
- The `NativeStakingSSVStrategy` contract in `NativeStakingSSVStrategy.sol` has an incorrect order of functions.

- The `ValidatorAccountant` contract in `ValidatorAccountant.sol` has an incorrect order of functions.
- The `ValidatorRegistrator` contract in `ValidatorRegistrator.sol` has an enum, which is a type declaration, declared after state variables.

To improve the overall code legibility, consider standardizing ordering throughout the codebase as recommended by the Solidity Style Guide (Order of Functions).

Update: *Resolved. The Origin Protocol team stated:*

> *Acknowledged but will not change at this late stage. Good to know for new contracts.*

N-02 Variables Are Initialized to Their Default Values

Throughout the codebase, some variables are being initialized with their default values:

- The `i` variable in `InitializableAbstractStrategy.sol`
- The `i` variable in `InitializableAbstractStrategy.sol`
- The `i` variable in `InitializableAbstractStrategy.sol`
- The `i` variable in `ValidatorRegistrator.sol`

To avoid wasting gas, consider not initializing variables with the same values that they would have by default upon being declared.

Update: *Resolved. The Origin Protocol team stated:*

> *Acknowledged but will not change. Most of the code base initializes the `i` variable in `for` loops so will leave it for consistency. Some auditors recommend being explicit about what the starting value is. Our preference is to leave it out as suggested but we prefer consistency over a tiny gas saving.*

N-03 Incremental Update Is Not Wrapped in an Unchecked Block

Since Solidity version 0.8.0, for any arithmetic operation, the compiler automatically checks for over- and underflows, which costs gas. Since it is highly unlikely that a positively incrementing variable will overflow within a loop, the increment can be wrapped into an `unchecked` block.

This applies to the following instances:

- In line 104 of `InitializableAbstractStrategy.sol`
- In line 122 of `InitializableAbstractStrategy.sol`
- In line 185 of `InitializableAbstractStrategy.sol`

Consider wrapping the increment inside an `unchecked` block to save the gas required to check against overflows.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *Acknowledged but will not change. We would prefer upgrading to Solidity 0.8.22 or higher which introduced an overflow check optimization that automatically generates an unchecked arithmetic increment of the counter of `for` loops. Upgrading the Solidity version is out of the scope of this project.*

N-04 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *Acknowledged but will not change. We can do this as a separate project.*

N-05 `require` Statement With Multiple Conditions

Throughout the codebase, there are `require` statements that require multiple conditions to be satisfied:

- The require statement in `InitializableAbstractStrategy.sol`
- The require statement in `ValidatorAccountant.sol`
- The require statement in `ValidatorAccountant.sol`
- The require statement in `ValidatorAccountant.sol`

To simplify the codebase and raise the most helpful error messages for failing `require` statements, consider having a single require statement per condition.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *Acknowledged but will not change. We think this is a good practice, especially for functions that are often called by users. But in these cases, they are in premissioned functions that would rarely be called. And if they are called, it will be via a Gnosis Safe so incorrect params will be detected before the tx is executed.*

N-06 Unnecessary Conditional

The `require` statement in `_doAccounting` checks that both `_fuseIntervalStart` and `_fuseIntervalEnd` are less than 32 `ether` and that `_fuseIntervalStart < _fuseIntervalEnd`. However, the `_fuseIntervalStart < 32 ether` condition is redundant as this condition is already ensured by the two other checks which ensure that `_fuseIntervalEnd` is already less than 32 `ether` and the start of the interval is less than the end.

Consider removing the `_fuseIntervalStart < 32 ether` condition.

Update: *Resolved in pull request #2082. The Origin Protocol team stated:*

> *Accepted and fixed.*

N-07 Lack of Indexed Event Parameters

Throughout the codebase, several events do not have indexed parameters.

To improve the ability of off-chain services to search and filter for specific events, consider indexing event parameters.

Update: *Resolved in pull request #2083. The Origin Protocol team stated:*

> *We have changed the easy ones with `address` type. Although `bytes` types can be indexed, the Waffle framework used for testing does not support it. As a compromise, we left the `pubKey` parameter unindexed and added a `bytes32 pubKeyHash` that is indexed. As the name suggests, this is a `keccak256` hash of the `pubKey`.*

Client Reported

CR-01 Beacon Chain Deposits Can Be Front-Run

The Beacon Chain permits multiple deposits for a single validator public key but does not verify whether they have the same withdrawal credentials. Therefore, a legitimate deposit to the validator could be front-run by an attacker setting a different withdrawal credential input than the original deposit. This exploit could only be performed by a user with full access to the validator private key which, in Origin's case, is their P2P staking partner.

Another possibility is that the registrator could call the stake function with an invalid signature which would cause ETH to be locked in the deposit contract. In addition, if an incorrect signature with a correctly recalculated `deposit_data_root` is passed to the deposit contract, the Beacon Chain would ignore that deposit. The registrator role is set by governance and there is only one registrator at a time which reduces the likelihood of such an attack.

Update: *Resolved in pull request #2074. The Origin Protocol team stated:*

> *Limits have been put in place to restrict the loss of funds from this type of attack.*

CR-02 Incorrect Deposit Event Emission

The amount in the `Deposit` event is incorrect if WETH is already present in the native staking contract.

Consider either checking if the ETH amount is a multiple of 32 or taking into account the fact that the event emission could be inaccurate.

Update: *Resolved. The Origin Protocol team stated:*

> *This has since been fixed.*

Recommendations

Monitoring Recommendations

When a large amount of slashing has occurred, a fuse will be triggered which pauses the contracts until the accounting is manually fixed. The fuse only triggers when the ETH modulo 32 `ether` is between the fuse start and end intervals. However, it is possible for so much slashing to occur that the ETH amount goes below the fuse start interval, in which case the accounting function would register a gain in consensus rewards rather than a loss due to slashing.

Consider monitoring the state of all the validators staking for Origin and constantly comparing the internal accounting to the real rewards and slashing amounts. Especially monitor for unexpected slashing events that would break the accounting to be able to react to them before they are swept into the strategy and wrongly accounted for.

Conclusion

During the audit, we found that there could be potential misalignments between the accounting of the contracts and the amount of rewards and slashing happening on the Beacon Chain. This is a trade-off of not using a staking oracle, which is known by the Origin team.

We would like to thank the Origin team for being responsive to our inquiries and providing extensive documentation. This aided our understanding of how the `NativeStakingSSVStrategy` contract interacted with the rest of the OETH ecosystem as well as the logic behind the unique reward and slashing accounting design. The system is thoughtfully designed but still depends on off-chain monitoring and manual intervention in some cases which could be prone to future human mistakes.