OpenZeppelin | security

OETH Withdrawal Queue Audit

ORIGIN

August 9, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-07-29 To 2024-08-01
Languages	Solidity

Total Issues	14 (8 resolved, 3 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	4 (1 resolved, 1 partially resolved)
Notes & Additional Information	8 (5 resolved, 2 partially resolved)

Scope

We audited the OriginProtocol/origin-dollar repository at commit eca6ffa.

In scope were the following files:

```
contracts/contracts
├── interfaces
│   └── IVault.sol
└── vault
    ├── OETHVaultAdmin.sol
    ├── OETHVaultCore.sol
    ├── VaultAdmin.sol
    ├── VaultCore.sol
    └── VaultStorage.sol
```

System Overview

Origin Ether (OETH) is a liquid ETH staking token. Yield is generated through various strategies, of which the most important is collecting rewards from staking ETH through the SSV Network. These strategies act as yield-bearing collateral for OETH and accumulate yield, which is then distributed to OETH holders via a rebasing mechanism. Users deposit WETH into the vault contract and receive newly minted OETH in return. The deposited WETH is allocated to different strategies to generate yield.

The new scope introduced several changes to the logic of the vault contracts. The main update includes the implementation of a withdrawal queue for withdrawing assets from the OETH Vault. This enhancement improves the system by reserving earned yield and assets received from deposits to cover withdrawals by prioritizing the withdrawal queue before allocating additional funds to the strategies.

Users willing to withdraw their assets initiate their withdrawal by means of a withdrawal request which effectively burns their OETH. After a waiting period of 10 minutes, and given that there is enough liquidity in the withdrawal queue, the user is able to claim their withdrawal request and receive WETH at a 1-to-1 ratio with their burned OETH. This queue mechanism helps avoid the need for withdrawing WETH from the strategies to cover users' withdrawal requests which could prevent actions like exiting a validator in the case of the native staking strategy.

To increase robustness, a portion of the WETH is reserved in the contract to cover withdrawals by means of a buffer, meaning that a certain amount will not be transferred to strategies even if there are no pending withdrawals in the queue. However, the strategist, via a 2-of-9 multisig, can override the preservation of the buffer amount if necessary.

In order to prevent yield theft, whenever OETH is minted and surpasses the configured rebase threshold, the dripper contract, which is responsible for collecting streamed rewards, is called to collect any accumulated rewards. Afterwards, the system calls the rebase function before actually minting OETH to the user.

Security Model and Trust Assumptions

During the audit, the following trust assumptions were made about the codebase:

- The governor and strategist are acting with honesty and integrity, having the best intentions for the protocol and its stakeholders.
- The strategies configured by the vault are functioning correctly and working as intended. Regarding the native staking strategy, the SSV network validators are trusted to perform properly, avoiding slashes and penalties.
- The harvester and dripper contracts are operating as expected, efficiently managing and distributing the yield earned by the strategies to the vault.
- The governor or strategist is diligently monitoring the withdrawal queue, ensuring that SSV validators are exited in a timely manner to release additional WETH whenever the execution and consensus rewards are insufficient to meet withdrawal requests.

Medium Severity

M-01 `_checkBalance` Returns an Incorrect Value During Insolvency

The `_checkBalance` function returns the balance of an asset held in the vault and all the strategies. If the requested asset is WETH, the amount of WETH reserved for the withdrawal queue is subtracted from this balance to reflect the correct amount of workable assets. In this specific case, the function returns the same result as the `_totalValue` function.

In the event that the vault becomes insolvent (e.g., during a mass slashing event) and multiple users are requesting to withdraw their OETH, the WETH in the withdrawal queue may exceed the total amount of workable assets. In this case, the function should return a balance of 0. However, it will actually return the amount of WETH in the vault and strategies without subtracting the WETH reserved for the withdrawal queue.

Consider returning 0 in case the amount of WETH reserved for the withdrawal queue exceeds the total amount of workable assets. In addition, since the `_checkBalance` function should return the same value as the `_totalValue` function if the asset is WETH, consider calling `_totalValue` in `_checkBalance` or vice versa.

Update: *Resolved in pull request #2166 at commit 2a6901a.*

M-02 Incorrect `__gap` Variable Decrement in `VaultStorage`

In the audited version of the contract, 4 slots were added to the storage of the `VaultStorage` contract:

- `dripper`, which occupies 1 slot.
- `withdrawalQueueMetadata`, which occupies 2 slots given the packing of the 4 `uint128` values.
- `withdrawalRequests`, which occupies 1 slot.

However, the `__gap` variable was only decreased by 3 slots from 49 to 46 instead of 4 slots. This discrepancy in the `__gap` value could lead to a storage collision if a contract upgrade

occurs. In addition, the starting point for the `__gap` variable count is not clearly indicated in the code, making it prone to errors that could lead to storage collisions in future versions.

Consider updating the `__gap` variable to the correct value to prevent potential storage collisions. Moreover, adding code comments that clarify where the count for the `__gap` variable starts would help prevent similar issues in the future.

Update: *Resolved in pull request #2167 at commit 44c7740.*

Low Severity

L-01 Incorrect Check Against `autoAllocateThreshold`

The `_mint` function auto-allocates deposited assets into the configured strategies in case the amount is above a certain `autoAllocateThreshold`. This action is preceded by adding liquidity to the withdrawal queue as filling the withdrawal queue has priority over allocating assets to the strategies. However, `autoAllocateThreshold` is checked against `_amount`, which may be more than the available assets to allocate in case some of these assets were used to fill the withdrawal queue.

Consider checking the `autoAllocateThreshold` against the correct amount of unallocated assets after taking into account the assets used to fill the withdrawal queue.

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

> *We will not change this. We can see the point being made but we do not think that the change should be done. The idea is that larger mints can afford to spend the extra gas on an allocation to default strategies if there is enough liquidity. It is true that now since we have the withdrawal queue, it is possible for all the deposited WETH to be reserved for the withdrawal queue. Hence, there is no WETH to allocate to the default strategy. However, this logic is already inside the `_allocate` function. It checks for the available WETH, and if nothing is available, it exits. We do not see any benefit in moving this logic up to the `_mint` function.*

L-02 Minting and Claiming Fails if Dripper Address Is Not Set

If the address of the `dripper` contract is not configured, all attempts to collect assets from the dripper contract will fail. This will cause the following public-facing functions to revert:

- `mint`, when it calls `_mint` which collects assets from the dripper before rebasing.
- `claimWithdrawal`, when it collects assets from the dripper in case there is not enough liquidity available in the withdrawal queue.
- `claimWithdrawals`, when it collects assets from the dripper similarly to `claimWithdrawal`.
- `addWithdrawalQueueLiquidity`, when it collects assets from the dripper before allocating WETH to the withdrawal queue.

Consider verifying that the address of the dripper contract is not equal to `address(0)` before calling the `collect` function.

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

> *This is a valid concern. If the Dripper contract is not configured, the OETH Vault will revert on the call to a zero address Dripper. Ideally, the OETH Vault will revert with an error message that developers can easily understand (e.g., `No Dripper config`). We do not think that we want to allow mints if the Dripper was misconfigured. The idea of calling the Dripper on large mints is to prevent yield attacks where someone can mint a large amount to collect most of the yield from the Dripper. This can happen if the daily job that collects the rewards from the Dripper is not running. If a month's worth of rewards is built up, the yield could be significant. Although anyone can call `collectAndRebase` on the Dripper, there is no incentive to do so. So, we would prefer the external functions like `mint` and `claimWithdrawal` to fail if the OETH Vault incorrectly configured the Dripper to a zero address. We do not think that we want to add extra logic to turn the failed call into a Vault-specific error message like `No Dripper config`.*

L-03 Missing Docstrings

Throughout the codebase, a few instances of missing docstrings were identified:

- The `IVault` interface in `IVault.sol`
- The `VaultStorage` contract in `VaultStorage.sol`
- The `VaultCore` contract in `VaultCore.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #2169 at commit a5ec05f.*

L-04 Incomplete Docstrings

Throughout the codebase, several instances of incomplete docstrings were identified where either parameters or return values were not fully documented:

- The `requestWithdrawal` function in `OETHVaultCore.sol`
- The `swapper` function in `VaultAdmin.sol`
- The `supportAsset` function in `VaultAdmin.sol`
- The `setMaxSupplyDiff` function in `VaultAdmin.sol`
- The `setTrusteeAddress` function in `VaultAdmin.sol`
- The `setTrusteeFeeBps` function in `VaultAdmin.sol`
- The `calculateRedeemOutputs` function in `VaultCore.sol`
- The `getAssetCount` function in `VaultCore.sol`
- The `getAssetConfig` function in `VaultCore.sol`
- The `getAllAssets` function in `VaultCore.sol`
- The `getStrategyCount` function in `VaultCore.sol`
- The `getAllStrategies` function in `VaultCore.sol`

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially resolved in pull request #2170 at commit f53193a.*

Notes & Additional Information

N-01 Use `calldata` Instead of `memory`

When dealing with function parameters in `external` functions, it is more efficient to use `calldata` instead of `memory`. `calldata` is a read-only region of memory that contains the function arguments, which makes it cheaper and more efficient compared to `memory`. Using `calldata` can save gas and improve the performance of a smart contract.

Within `OETHVaultCore.sol`, the `_requestIds` parameter should use `calldata` instead of `memory`.

Consider using `calldata` for function parameters in `external` functions to optimize gas usage.

Update: *Resolved in pull request #2171 at commit 26b1458.*

N-02 Unnecessary Cast

Within the `VaultAdmin` contract, the `address(_fromAsset)` cast is unnecessary.

To improve code clarity, consider removing any unnecessary casts.

Update: *Resolved in pull request #2172 at commit 8426bc2.*

N-03 Unused Named Return Variables

Named return variables are a way to declare variables that are meant to be used within a function's body for the purpose of being returned as that function's output. They are an alternative to explicit in-line `return` statements.

Throughout the codebase, a few instances of unused named return variables were identified:

- The `amount` return variable in the `_claimWithdrawal` function in `OETHVaultCore.sol`
- The `value` return variable in the `_totalValue` function in `OETHVaultCore.sol`

- The `value` return variable in the `_totalValue` function in `VaultCore.sol`

Consider either using or removing any unused named return variables.

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

> *Although we don't explicitly assign the return variables, we think it still useful for documentation purposes to define the return variables.*

N-04 Incorrect or Misleading Documentation

Throughout the codebase, a few instances of incorrect or misleading documentation were identified:

- The documentation of the `requestWithdrawal` function states that there is no minimum time or block number before a request can be claimed. However, there is in fact a minimum time of 10 minutes before a request can be claimed.
- The NatSpec of the `requestWithdrawal` function states that the function has three parameters. However, the function only has a single `_amount` parameter and two return variables `requestId` and `queued`.
- The documentation of the `_checkBalance` function states that it will only return a non-zero balance for WETH. However, it could return 0 in case the total amount of assets in the vault and the strategies is equal to the amount of assets requested for withdrawal. In addition, it should return 0 in case the total amount of assets in the vault and the strategies is equal to the amount of assets requested for withdrawal as described in issue `M-01`.
- The documentation of the `_checkBalance` function states that all assets, except for WETH, will return 0 even if there is some dust amount left in the Vault. However, this function calls the `_checkBalance` function of the inherited `VaultCore` contract, which could return a non-zero balance for assets other than WETH.

Clean and accurate documentation helps users and developers better understand the codebase. As such, consider updating the identified instances of incorrect or misleading documentation.

Update: *Resolved in pull request #2173 at commit ed1f74f.*

N-05 State Variable Visibility Not Explicitly Declared

Throughout the codebase, several instances of state variables lacking an explicitly declared visibility were identified:

- The `CLAIM_DELAY` state variable in `OETHVaultCore.sol`
- The `adminImplPosition` state variable in `VaultStorage.sol`
- The `MINT_MINIMUM_UNIT_PRICE` state variable in `VaultStorage.sol`
- The `MIN_UNIT_PRICE_DRIFT` state variable in `VaultStorage.sol`
- The `MAX_UNIT_PRICE_DRIFT` state variable in `VaultStorage.sol`

For better code clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

Update: *Partially resolved in pull request #2174 at commit ee971c3. The Origin Protocol team stated:*

> *We have changed the first one. The others are long-existing variables so we have left them for now.*

N-06 Missing Named Parameters in Mappings

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

Within `VaultStorage.sol`, a few instances of mappings without named parameters were identified:

- The `assets` state variable
- The `strategies` state variable
- The `assetDefaultStrategies` state variable
- The `withdrawalRequests` state variable

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

Update: *Acknowledged, will resolve. The Origin Protocol team stated:*

> *We are going to leave it for now. However, in the new retro, we will raise the issue of whether we should adopt named parameters in mappings in the Origin dollar repo.*

N-07 Unused State Variable

Unused state variables can affect code readability and make it difficult to understand.

Within the `VaultCore` contract, the `MAX_UINT` state variable is unused.

To improve the overall clarity and readability of the codebase, consider removing any unused state variables.

Update: *Resolved in pull request #2175 at commit 682048a.*

N-08 Variables Are Initialized to Their Default Values

Throughout the codebase, a few instances of variables being initialized to their default values were identified:

- The `i` variable in `OETHVaultCore.sol`
- The `i` variable in `OETHVaultCore.sol`
- The `y` variable in `VaultCore.sol`
- The `ousdMetaStrategy` variable in `VaultStorage.sol`
- The `netOusdMintedForStrategy` variable in `VaultStorage.sol`
- The `netOusdMintForStrategyThreshold` variable in `VaultStorage.sol`

To avoid wasting gas, consider not initializing variables to the same values they would have by default when they are being declared.

Update: *Partially resolved in pull request #2176 at commit 84fdd11. The Origin Protocol team stated:*

> *The first three have been changed. The last three were changed in commit `e464b7a8a6c9a4949030235e24666ab9fa770e45`. The only reason they were being initialized was to suppress Slither errors.*

Conclusion

The audited version of the OETH vault introduces the concept of a withdrawal queue to prioritize user withdrawals over allocating additional funds to the different configured strategies.

Several medium- and lower-severity vulnerabilities were found while recommendations aimed at improving the codebase were also made. The auditors appreciated the Origin Protocol team's responsiveness and eagerness to secure their system. Throughout the audit period, communication with the Origin Protocol team was great and all questions were answered promptly and thoroughly.

In addition to resolving the identified issues, we recommended monitoring the withdrawal queue to ensure that the yield generated by the different strategies is sufficient to cover the outstanding withdrawals. In case this would not be sufficient, liquidity should be exited from the strategies by the governor or strategist to cover the outstanding withdrawals.