OpenZeppelin | security

Plume Rooster AMO Audit

ORIGIN

July 3, 2025

Table of Contents

Summary

Type	DeFi	Total Issues	14 (8 resolved)
Timeline	From 2025-06-09 To 2025-06-13	Critical Severity Issues	0 (0 resolved)
Languages	Solidity	High Severity Issues	0 (0 resolved)
		Medium Severity Issues	0 (0 resolved)
		Low Severity Issues	5 (3 resolved)
		Notes & Additional Information	8 (4 resolved)
		Client Reported Issues	1 (1 resolved)

Scope

We audited pull request #2499 in the OriginProtocol/origin-dollar repository at the 2d02162 commit.

In scope was the `contracts/contracts/strategies/plume/RoosterAMOStrategy.sol` file.

System Overview

Pull request #2499 introduces the Plume Rooster AMO (automated market operations) vault strategy—an advanced DeFi liquidity mechanism designed to deposit funds into an underlying Rooster (Maverick) pool. This strategy will be deployed on the Plume network and is intended to support the superOETHp token launched by Origin Protocol. It deposits user-supplied WETH into the Rooster OETH/WETH pool, generating yield through Rooster token rewards. The strategy is designed to maximize returns via active liquidity management while mitigating risk through role-based controls and safety mechanisms.

To maintain the OETH peg with WETH, the strategy periodically rebalances the pool to ensure WETH is priced slightly higher than OETH. This enforced premium reflects WETH's typically higher liquidity and lower risk profile relative to OETH.

Integration with the Origin Vault restricts deposit and withdrawal functionality exclusively to the vault, preventing direct access by external users.

Security Model and Trust Assumptions

During the audit, the following trust assumptions were made regarding the in-scope code and the broader security model:

- The Rooster Protocol functions as documented and will not be upgraded to introduce breaking changes.
- The Maverick pool functions as documented and will not be upgraded to introduce breaking changes.
- The strategist will perform regular rebalancing of the pool.
- The WETH/OETH price remains relatively stable and near parity.
- The strategy contract will be used exclusively in combination with the Origin Vault.
- The governor will correctly and promptly call the `mintInitialPosition` and `setAllowedPoolWethShareInterval` functions following deployment to activate the strategy.

- The `_updateUnderlyingAssets` function relies on `sqrtPriceAtParity` being set to `sqrtPriceTickHigher` during deployment of the `RoosterAMOStrategy` implementation contract.

Privileged Roles

The following roles were identified within the audit scope and are assumed to operate in a trusted capacity:

- **Governor**: Authorized to perform administrative actions, including initial configuration and strategy setup.
- **Strategist**: Responsible for regularly rebalancing the pool to maintain intended exposure.
- **Vault**: Authorized to perform deposits and withdrawals with the strategy.
- **Harvester**: Responsible for collecting rewards and forwarding them to the vault.

Low Severity

L-01 `_checkForExpectedPoolPrice` Returns Incorrect WETH Share When Pool Is Out of Bounds

The `_checkForExpectedPoolPrice` function checks whether the Maverick pool's current price is within a preconfigured tick range and returns the corresponding WETH share held by the strategy. While the price check itself functions correctly, the returned `wethSharePct` is inaccurate when the pool price is outside the expected range.

Specifically, the function returns a WETH share of `0` in all out-of-bounds cases. While this is correct when the price is at or below the lower tick (where the position is 100% OETH), it is incorrect when the price is at or above the upper tick boundary. In that scenario, the position is entirely in WETH, and the WETH share should be `1e18`.

Although the current implementation does not rely on the returned WETH share value of `0`, this behavior introduces a latent risk. Because the contract is upgradeable, future versions of the strategy may assume `wethSharePct` is accurate and make faulty decisions based on it.

Consider updating the logic to return the correct WETH share based on whether the price is below the lower tick (`0`) or above the upper tick (`1e18`). This change would make the strategy more robust and reduce the risk of incorrect assumptions in future upgrades or extensions.

Update: *Resolved in pull request #2542.*

L-02 Incorrect Liquidity Calculation in `_getAddLiquidityParams` When One Token Is Not Required

The `_getAddLiquidityParams` function calculates the amount of liquidity to be added based on available WETH and OETH balances. In cases where the tick lies at the far ends of the curve, the pool may only require one of the two tokens. When this happens, the function sets the required amount for the unused token to `1` to avoid a division-by-zero error.

This behavior introduces a silent correctness issue. By forcibly setting `WETHRequired` or `OETHRequired` to `1`, the result of the `min(...)` operation used to compute liquidity can be incorrectly limited by a token that is not actually needed (e.g., limited by WETH when `WETHRequired == 0`). This becomes particularly problematic when the available balance of the unused token is significantly smaller than that of the required token.

This edge case arises when the position is entirely in a single token—either WETH or OETH—depending on the tick location. Instead of masking the zero by setting it to `1`, the calculation should explicitly handle these cases. If only one token is required, the liquidity should be calculated solely based on that token. If both `WETHRequired` and `OETHRequired` are zero, the function should revert, as this likely indicates a deeper issue with tick selection or quoter behavior.

Fixing this would ensure accurate liquidity scaling and improve capital efficiency in edge pool configurations.

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

> *The strategy contract has a hardcoded limit that does not allow deposits below 1% and above 95% of the WETH share. For this reason, the mentioned edge cases are not reachable. We could correctly handle these situations when constructing liquidity parameters, but we feel that it would introduce additional complexity that is not justified due to handling an unreachable situation.*

L-03 Incomplete Docstrings

Within `RoosterAMOStrategy.sol`, multiple instances of incomplete docstrings were identified:

- The return value is not documented in the `supportsAsset` function.
- The return value is not documented in the `tickDominance` function.

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

Update: *Resolved in pull request #2545. The Origin team stated:*

> *We have fixed the issue as suggested.*

L-04 Missing Docstrings

Within `RoosterAMOStrategy.sol`, multiple instances of missing docstrings were identified:

- The `PoolWethShareIntervalUpdated` event
- The `LiquidityRemoved` event
- The `PoolRebalanced` event
- The `UnderlyingAssetsUpdated` event
- The `LiquidityAdded` event

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

Update: *Resolved in pull request #2546. The Origin team stated:*

> *We have fixed the issue as suggested.*

L-05 Require Statement With Multiple Conditions

Within `RoosterAMOStrategy.sol`, the `require(allowedWethShareStart != 0 && allowedWethShareEnd != 0, "Weth share interval not set")` requires multiple conditions to be satisfied.

To simplify the codebase and produce clearer error messages for failing `require` statements, consider separating this into individual `require` checks for each condition.

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

> *We agree that it is normally preferable to have a separate `require` statement for each condition. However, in this case, both variables are set at the same time. For this reason, we feel that there is no need to have separate `require` statements.*

Notes & Additional Information

N-01 Typographical Errors

The following typographical errors and minor formatting issues were found in the `RoosterAMOStrategy` contract:

- Line 71: "provider" should be "provide"
- Line 268: "withing" should be "within"
- Line 971: "99,8 %" should be "99.8 %"
- Line 986: "Collect stkAave, convert it to AAVE send to Vault" should be "Collect Rooster reward tokens"
- Line 1129: "Approve all tokens" revert message should be updated to "Unsupported" for accuracy

Consider fixing these instances to improve the clarity and consistency of the codebase.

Update: *Resolved in pull request #2547. The Origin team stated:*

> *We have fixed the issue as suggested.*

N-02 Overly Restrictive Validation

Throughout the codebase, there are several validation issues that can be corrected:

- The NatSpec comment for `ACTION_THRESHOLD` describes it as the "Minimum liquidity required to continue," which implies the value `1e12` is an inclusive lower bound for an action to proceed. However, within the `_burnOethOnTheContract` function, this threshold must be exceeded, making the threshold an exclusive boundary and contradicting the NatSpec comment. Consider updating the NatSpec comment to clarify that the amount must be "greater than" the threshold to align with the code's logic.
- The validation checks in `_getAddLiquidityParams` use a strict greater-than (>) comparison to ensure the calculated required token amounts are less than the maximum available balances. This logic is overly restrictive and will incorrectly cause the transaction to revert in valid scenarios where the required amount is exactly equal to the contract's available balance. To fix this and allow the strategy to correctly deploy its full

token balance when necessary, consider changing the comparison operators from `>` to `>=`.

Consider fixing these issues to ensure the validations are correctly aligned with intended behavior.

Update: *Resolved in pull request #2548. The Origin team stated:*

> *We have fixed the issue as suggested.*

N-03 Unused Error

In `RoosterAMOStrategy.sol`, the `InsufficientTokenBalance` error is unused.

To improve the overall clarity, intentionality, and readability of the codebase, consider either using or removing any currently unused errors.

Update: *Resolved in pull request #2549.*

The unused `InsufficientTokenBalance` error has been removed. The Origin team stated:

> *We have fixed the issue as suggested.*

N-04 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.

Within `RoosterAMOStrategy.sol`, multiple instances of unused named return variables were identified:

- The `_isExpectedRange` return variable of the `_checkForExpectedPoolPrice` function
- The `_wethSharePct` return variable of the `_checkForExpectedPoolPrice` function
- The `_amountWeth` return variable of the `getPositionPrincipal` function
- The `_amountOeth` return variable of the `getPositionPrincipal` function
- The `_tickDominance` return variable of the `tickDominance` function

Consider either using or removing any unused named return variables to improve clarity and reduce potential confusion in the codebase.

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

> *We have decided to keep it as it is within all three functions, as we believe that this way is better for code readability. Named variables make it easier to understand the meaning of return variables. The early return statements would expand to multiple lines, which, in our opinion, makes the code less readable.*

N-05 Inconsistent Use of Returns in a Function

To improve the readability of the function, use the same return style.

Within `RoosterAMOStrategy.sol`, there are multiple instances where functions have inconsistent usage of returns.

- In `_checkForExpectedPoolPrice function`.
- In `getPositionPrincipal function`.
- In `tickDominance function`.

Consider being consistent with the use of returns throughout the functions.

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

> *We have decided to keep it as it is because we believe that changes will worsen code readability.*

N-06 Use Custom Errors

Since Solidity version 0.8.4, custom errors provide a cleaner and more cost-efficient way to explain to users why an operation failed.

Throughout the `RoosterAMOStrategy.sol` file, multiple instances of `revert` and `require` messages were found that contain error messages as strings.

For conciseness and gas savings, consider using custom errors throughout the codebase.

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

> *Indeed, we do have error inconsistencies across the codebase and will fix them sometime in the future.*

N-07 Custom Errors in `require` Statements

Since Solidity version `0.8.26`, custom error support has been added to `require` statements. While initially this feature was only available through the IR pipeline, Solidity `0.8.27` extended support to the legacy pipeline as well.

The `RoosterAMOStrategy.sol` contains multiple instances of `if-revert` that could be replaced with `require` statement:

- The `if (_throwException) { revert OutsideExpectedTickRange(); }` statement.
- The `if (_throwException) { revert PoolRebalanceOutOfBounds(_wethSharePct, allowedWethShareStart, allowedWethShareEnd); }` statement.
- The `if (_swapWeth) { revert NotEnoughWethForSwap(_balance, _amountToSwap); }` statement.
- The `if (amountOut < _minTokenReceived) { revert SlippageCheck(amountOut); }` statement.
- The `if (_wethInThePool < _additionalWethRequired) { revert NotEnoughWethLiquidity(_wethInThePool, _additionalWethRequired); }` statement.

For conciseness and gas savings, consider replacing `if-revert` statements with `require` ones.

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

> *Implementing this change would require increasing the Solidity version across the codebase. That is a more involved change that we will do sometime in the future.*

N-08 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.

The `RoosterAMOStrategy` contract does 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: *Resolved in pull request #2551. The Origin team stated:*

> *We have fixed the issue as suggested.*

Client Reported

CR-01 Corrected Rounding for WETH Share Removal

The share removal calculation in `_ensureWETHBalance` was not precisely accurate due to various rounding effects in the Maverick pool.

This could cause the strategy to underestimate the amount of liquidity to remove, occasionally resulting in insufficient WETH being made available when needed.

The Origin team identified this issue during fuzz testing and corrected it in commit a4f252b by adjusting the calculation to account for edge cases. The updated logic ensures the strategy removes a minimally sufficient amount of liquidity.

Update: *Resolved in pull request #2499.*

Conclusion

The Plume Rooster AMO strategy introduces a sophisticated liquidity management mechanism aimed at optimizing yield while maintaining price alignment between OETH and WETH in the Maverick V2 pool on Plume. By enforcing a slight price premium for WETH and limiting asset interactions to the Origin Vault, the strategy seeks to balance risk, return, and protocol safety through controlled access, regular rebalancing, and predefined solvency thresholds. Its design reflects a thoughtful tradeoff between performance and security, supported by role-based governance.

During the review, only low and informational severity issues were identified, reflecting a generally well-structured codebase and security-conscious development process. All findings have been addressed to a satisfactory extent, contributing positively to the overall security and robustness of the system.

The Origin team was highly responsive and collaborative throughout the audit process, and their professionalism is greatly appreciated.