OpenZeppelin | security

Sonic SwapX AMO Strategy Audit

ORIGIN

April 11, 2025

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 5

Security Model and Trust Assumptions 5

Privileged Roles 6

Medium Severity 8

M-01 Gauge Emergency Functions Uncallable 8

Low Severity 8

L-01 Leftover Tokens From Skimming Pools 8

L-02 External Calls to gaugeRewarder in Gauge Contract 9

L-03 Redundant Check in _swapExactTokensForTokens 10

Notes & Additional Information 10

N-01 Unnecessary Boolean Literal 10

N-02 Unnecessary Casts 11

Client Reported 11

CR-01 Deposits Can Be Front-Run 11

Conclusion 13

Summary

Type	DeFi/Stablecoin
Timeline	From 2025-03-20 To 2025-03-28
Languages	Solidity

Total Issues	7 (2 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	3 (0 resolved)
Notes & Additional Information	2 (0 resolved)
Client Reported Issues	1 (1 resolved)

Scope

We audited the OriginProtocol/origin-dollar repository at commit 8a1df95.

In scope was the following file:

```
contracts/contracts/strategies/sonic/SonicSwapXAMOStrategy.sol
```

System Overview

The `SonicSwapXAMOStrategy.sol` contract is intended to be deployed on the Sonic chain, and will interact with the SwapX stable pool with Origin Sonic (OS) and wrapped Sonic (wS) tokens. It can mint and burn OS tokens up to a solvency check to deposit into or withdraw from the pool, and can also rebalance the pool when the reserve ratio goes out of balance. The value of this strategy will be held in its LpTokens when assets are deposited into the pool. The lpTokens are further put into the SwapX gauge to earn rewards that are regularly harvested by a designated harvester.

This strategy will be connected to the `OSonicVault` on the Sonic chain. Hence, the profit from the strategy contributes to the total value of the vault, which rebalances the OSonic token value.

Security Model and Trust Assumptions

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

- The `SonicSwapXAMOStrategy` contract mostly contains permissioned functions. The liquidity actions interacting with the connected SwapX pool can either come from the vault or the strategist of the vault. It is not possible to be triggered by a non-privileged user from the vault if the `SonicSwapXAMOStrategy` is not set as the vault's default strategy. Primarily, the governor will set most of the strategy parameters.
- By interacting with SwapX stable pool and leaving the entire lpToken balance with SwapX gauge, the Origin team is trusting that the SwapX pool as well as the gauge are secure and working as expected. The entire value of the strategy will, most of the time, be held in the gauge contract. The assumptions surrounding the SwapX gauge, if violated, may result in the locking of funds. Furthermore, `SonicSwapXAMOStrategy` is an upgradeable contract and, thus, the implementation can be upgraded by the governor.

- The allowed strategists of the vault will use MEV protections such as the VaultValueChecker to wrap all liquidity interactions with the pool so that the change in vault value is ensured to be within a certain tolerance. Thus, this acts as general slippage protection for interacting with the pool under uncertain on-chain conditions.

Privileged Roles

Multiple privileged roles were identified in the logic of the `SonicSwapXAMOStrategy` contract.

`Governor`

`Governor` is a core role of the Origin protocol that holds the highest level of privileges. In the `SonicSwapXAMOStrategy` contract, it can perform several highly sensitive operations. Specifically, the `Governor` role can:

- Call the `initialize` function during deployment.
- Call the `removePToken` and `setPTokenAddress` functions to manage the `assetToPToken` mapping and `assetsMapped` array. These entries are unused in practice, as the strategy relies on immutable variables to store the `sAMM-wS/OS` pool and wS addresses.
- Call the `safeApproveAllTokens` function to approve the full balance of `sAMM-ws/OS` LP tokens to the gauge.
- Call the `setHarvesterAddress` function to define the account allowed to collect the accumulated `SWPx` from the gauge.
- Call the `transferToken` function to move any ERC-20 tokens held by the strategy.
- Call the `transferGovernance` function to initiate a change of the governor account.

`Strategist`

The `Strategist` role is a core role in the Origin protocol, albeit with fewer privileges than the `Governor` role. In the `SonicSwapXAMOStrategy` contract, the `Strategist` role can:

- Call `swapAssetsToPool` when the pool is imbalanced towards OS. This removes liquidity from the pool, swaps the wS removed to OS, and burns all the resulting OS tokens.
- Call `swapOTokensToPool` when the pool is imbalanced towards wS tokens. This mints new OS tokens, swaps them into the pool, then mints additional OS tokens and deposits them together with the wS tokens resulting from the swap as liquidity.

`Vault`

The `Vault` role in `SonicSwapXAMOSStrategy` restricts the various entry points to be called only by the `OSonicVaultCore` contract. However, the functions within `OSonicVaultCore` that trigger these entry points can only be called by either the `Governor` or the `Strategist`, except for `OSonicVaultCore.removeStrategy`, which is exclusively callable by the `Governor`:

- `deposit`: deposits a specified amount of wS into the pool.
- `depositAll`: deposits the entire wS balance of the strategy into the pool.
- `withdraw`: withdraws a specified amount of wS from the pool.
- `withdrawAll`: withdraws the full value of wS deposited in the pool.

`Harvester`

`Harvester` is the only role that is allowed to call `collectRewardTokens` to collect the Strategy's `SWPx` rewards from the gauge and send them to itself.

SwapX Gauge Owner

The LP tokens are typically held in the Swap X gauge, which is an immutable contract. However, it is important to note that the gauge has an `Owner` role that has the permission to perform certain trusted operations within the gauge. Specifically, the `Owner` can:

- Call `setDistribution` to assign a distribution address.
- Call `setGaugeRewarder` to set the gauge rewarder address.
- Call `setInternalBribe` to configure the internal bribe address.
- Call `activateEmergencyMode` and `stopEmergencyMode` to toggle the emergency mode on or off.

Medium Severity

M-01 Gauge Emergency Functions Uncallable

The SwapX Gauge contract has an emergency switch. While this switch can be toggled between `true` and `false`, setting it to `true` will prevent withdrawals and deposits. Instead, there are special `emergencyWithdraw` and `emergencyWithdrawAmount` functions for use during emergencies.

Consider implementing an emergency withdrawal function within the `SonicSwapXAMOStrategy` contract that checks if the Gauge is in "emergency" mode and calls the emergency withdrawal functions. Alternatively, consider implementing such a check and an emergency withdraw branch in the existing withdrawal functions.

Update: *Resolved in pull request #2463 at commit 2e2a3d6. The Origin team stated:*

> *Thanks for finding this one. This issue was fixed when it was raised in Slack.*

Low Severity

L-01 Leftover Tokens From Skimming Pools

The `deposit` and `depositAll` functions first skim the pool, thus potentially obtaining OS and wS token balances in the strategy contract. The extra wS amount is taken care of in the `depositAll` function. However, the extra OS amount is not accounted for when computing the `osDepositAmount` value to be minted. Thus, it is possible to have a leftover OS token balance in the strategy contract after the deposit calls.

Consider taking into account the current OS and wS balance when computing the `osDepositAmount` value. Alternatively, consider documenting the expectation of leftover token amounts from the skimming pool for deposit calls.

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

> *We think this is ok. The vault uses the strategy's `depositAll` function in it's `depositToStrategy` function. `withdrawAll` will also pick up any extra OS tokens in the strategy.*

L-02 External Calls to `gaugeRewarder` in Gauge Contract

The `GaugeV2` contract makes frequent calls to the `onReward` function of `gaugeRewarder`, but only when the `gaugeRewarder` address is not zero. Currently, the `gaugeRewarder` address is 0. However, at any point, the owner may call `setGaugeRewarder` to change the `gaugeRewarder` address. There could be two implications when that happens.

Firstly, if the `gaugeRewarder.onReward` function reverts for some reason, it will cause calls to the `GaugeV2` contract's `deposit`, `withdraw`, and `getReward` functions to fail, resulting in an inability to successfully execute many functions within the `SonicSwapXAMOStrategy` contract. This can lead to funds being locked in the `GaugeV2` contract.

Secondly, if the strategy's `checkBalance` function is called within the `gaugeRewarder.onReward` hook, the value may not be correct. This is because, in the middle of the withdrawal, the LP tokens are burned from the gauge balance but not yet transferred to the strategy. The `checkBalance` function only accounts for the strategy's wS token balance and the LP tokens deposited in the gauge. As a result, during that external call, the `checkBalance` function will report an incorrect balance.

All functions that use `checkBalance` in `Vault` are `nonReentrant`, so, reentrancy is not possible there. However, it may leave a chance for third-party protocols interacting with the vault to see an incorrect value when calling `VaultCore.totalValue()` or `VaultCore.checkBalance()` from the `gaugeRewarder.onReward` hook.

Considering the potential impact of having funds locked in the gauge, consider monitoring the `GaugeV2` contract's `gaugeRewarder` state variable for any changes. If possible, simulate the execution of the `onReward` function. In addition, consider removing liquidity before changes to the `gaugeRewarder` are made and evaluating the `gaugeRewarder` before re-depositing.

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

> *The OS/wS gauge is owned by `GaugeFactoryV2` which is owned by the 2/6 Gnosis Safe `0xAdB5A1518713095C39dBcA08Da6656af7249Dd20`. Funds can be*

> *withdrawn by the `emergencyWithdraw` function if `gaugeRewarder` is misconfigured. It wraps the call to `onReward` in a `try/catch` block.*

L-03 Redundant Check in `_swapExactTokensForTokens`

Within the `_swapExactTokensForTokens` function, the check on `_tokenIn` and `_tokenOut` is not needed. This is because `_swapExactTokensForTokens` is `internal`, and at both places where it is called, it is already ensured that this check is valid.

Consider removing the aforementioned redundant check.

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

> *We agree that this is not needed, but gas is not an issue on Sonic. So, we can be extra careful with checks. The extra checks were more for covering a future scenario where `_swapExactTokensForTokens` was called by something else.*

Notes & Additional Information

N-01 Unnecessary Boolean Literal

In this `require` statement, the use of the boolean literal is unnecessary. The statement can simply be `IPair(_baseConfig.platformAddress).isStable()` instead of `IPair(_baseConfig.platformAddress).isStable() == true`.

Consider removing the boolean literal within the conditional statement to make the code clearer and more concise.

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

> *We think that it is clearer to see the explicit comparison to a boolean and not a number. Although, it is less concise. We will leave this as it is.*

N-02 Unnecessary Casts

Within the `SonicSwapXAMOStrategy` contract, the `address(gauge)` cast is unnecessary.

To improve the overall clarity and intent of the codebase, consider removing any unnecessary casts.

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

> *The first `address(gauge)` cast is unnecessary. But we will leave it as it is.*

Client Reported

CR-01 Deposits Can Be Front-Run

Since LP tokens are always minted at the pool ratio (computed atomically in the strategy, with the pool skimmed beforehand), the amount of LP tokens minted is:

$$\frac{wsAmount \times LPtokenTotalSupply}{wsReserves}$$

An attacker can directly manipulate the `wsReserves` value by swapping. The exploit flow is as follows:

1. Attacker mints LP tokens.
2. Attacker swaps wS → OS, devaluing wS in the pool.
3. Strategist deposits wS at this fake, discounted price. As a result, the amount of LP tokens received is lower than expected. The price remains unchanged.
4. Attacker swaps back OS → wS, restoring the original price.
5. Attacker burns the LP tokens.

The Strategist mints liquidity at a manipulated price, receiving an unfair amount of LP tokens. The wS deposited would be worth more LP tokens otherwise. That lost value is captured by the attacker's LP tokens.

A separate pull request has added an explicit slippage mechanism to the strategy contract to prevent pool balance manipulation.

Update: *Resolved in pull request #2464 at commit 92aad92. The Origin team stated:*

> *We went with a solution that limits the damage by restricting deposits to the pool when the pool's OS/wS price is within 1% (above or below) of the peg.*

Conclusion

The Origin team was highly engaged in supporting the security researchers, providing clear guidance on the audit scope and broader system context. The codebase demonstrated a clean and thoughtful design, reflecting careful implementation. Although certain trust assumptions rely on the SWAPx team, the code itself appears robust. No major issues were identified; however, several suggestions were offered to address edge cases and enhance overall code quality.