STATE MIND

Symbiotic Hooks

22-12-2024 - 27-12-2024

Table of contents

1. Project brief

Title	Description
Client	Symbiotic
Project name	Symbiotic Hooks
Timeline	22-12-2024 - 27-12-2024

Project Log

Date	Commit Hash	Note
25-12-2024	2d754305ad3bf26ca09b3a3ca456f774f485f309	Initial Commit
05-01-2025	1a873e1975ed675e4ab6327b292fc980b1ec8bcc	Reaudit

Short Overview

Symbiotic Hooks is a complimentary system for handling slashing actions within Symbiotic Core. Symbiotic Delegator modules don't update allocations during slashing events in any way. The logic for handling slashes has been transferred to hooks smart contracts, which can not only modify stake allocations in delegators but also be part of a system built on top of core contracts, altering its state and various parameters. In the basic implementation of hooks, there are several behavior options:

- Decrease Hook — this hook decreases the stake allocation in the corresponding delegator by the slash amount.
- Reset Hook — tracks the number of slashes, and after a certain number within a specific period, it resets the entire stake.
- Redistribute Hook — a hook with special logic for NetworkRestakeDelegator, which decreases the stake of the slashed operator and redistributes it among other operators.

Project Scope

The audit covered the following files:

- NetworkRestakeDecreaseHook.sol
- NetworkRestakeRedistributeHook.sol
- FullRestakeResetHook.sol
- NetworkRestakeResetHook.sol
- OperatorSpecificResetHook.sol
- FullRestakeDecreaseHook.sol
- OperatorSpecificDecreaseHook.sol

2. Finding severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Client regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Client is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	1 (1 fixed, 0 acknowledged)
Medium	1 (1 fixed, 0 acknowledged)
Informational	5 (3 fixed, 2 acknowledged)
Total	7 (5 fixed, 2 acknowledged)

4. Conclusion

During the audit of the codebase, 7 issues were found in total:

- 1 high severity issue (1 fixed)
- 1 medium severity issue (1 fixed)
- 5 informational severity issues (3 fixed, 2 acknowledged)

The final reviewed commit is 1a873e1975ed675e4ab6327b292fc980b1ec8bcc

5. Findings report

HIGH-01	Subnetwork slash overlap caused by a single buffer	Fixed at: 3305e47

Description

Lines:

- FullRestakeResetHook.sol#L27
- NetworkRestakeResetHook.sol#L27
- OperatorSpecificResetHook.sol#L27

In the OperatorSpecificResetHook, the number of operator slashes is counted for each vault in which they have opted in. However, an operator might have multiple subnetworks within the same vault. Since the buffer is created for the entire vault rather than for the vault–subnetwork pair, slashes in one subnetwork can cause the limits in another subnetwork to be reset. The same issue occurs in other reset hooks as well.

Recommendation

We recommend adding separate buffers not for a single entity, but for entity pairings (vault–subnetwork, vault–subnetwork–operator).

MEDIUM-01	Missing buffer update	Fixed at: 26da5c1

Description

Lines:

- OperatorSpecificResetHook.sol#L70
- NetworkRestakeResetHook.sol#L71
- FullRestakeResetHook.sol#L71

The reset hooks contracts don't handle cases where an operator's limits are set to 0 after **SLASH_COUNT** slashes but then increase again. The buffer contains old slashings and **_slashings[vault][operator].count()** will always return **SLASH_COUNT**. To handle the situation, we will need to redeploy the hook contract, as we don't have any options to clear the buffer manually. Also, according to OpenZeppelin documentation, **.length()** method should be used during initial setup in all reset hook contracts instead of **.count()**, to save gas on re-initialization:

```
if (_slashings[vault][operator].length() == 0) {
    _slashings[vault][operator].setup(SLASH_COUNT);
}
```

Recommendation

We recommend clearing **_slashings** buffer after the limit reset and using **.length()** method.
For example, **NetworkRestakeResetHook.sol**:

```
if (
    _slashings[vault][operator][subnetwork].count() == SLASH_COUNT
        && Time.timestamp() - uint256(_slashings[vault][operator][subnetwork].last(SLASH_COUNT - 1)) <= PERIOD
) {
    INetworkRestakeDelegator(msg.sender).setOperatorNetworkShares(subnetwork, operator, 0);
    _slashings[vault][operator].clear(); // clear buffer
}
```

INFORMATIONAL-01	Initialize circular buffer only if it will be used	Fixed at: 00e60be

Description

Lines:

- FullRestakeResetHook.sol#L60
- NetworkRestakeResetHook.sol#L60
- OperatorSpecificResetHook.sol#L59

In the reset hook contracts, a check is performed to determine if the circular buffer is initialized before data is pushed into it. However, the function may exit before the data is pushed, rendering the buffer initialization check redundant.

Recommendation

We recommend performing the buffer setup check after verifying the network limit. This avoids redundant setup and saves gas in cases where the function returns prematurely:

```
- if (_slashings[vault].count() == 0) {
-     _slashings[vault].setup(SLASH_COUNT);
- }
- if (IOperatorSpecificDelegator(msg.sender).networkLimit(subnetwork) == 0) {
-     return;
- }
+ if (IOperatorSpecificDelegator(msg.sender).networkLimit(subnetwork) == 0) {
+     return;
+ }
+ if (_slashings[vault].count() == 0) {
+     _slashings[vault].setup(SLASH_COUNT);
+ }
_slashings[vault].push(bytes32(uint256(Time.timestamp())));
```

INFORMATIONAL-02	Fixed slash conditions in reset hooks allow unpenalized malicious actions	Acknowledged

Description

Lines:

- OperatorSpecificResetHook.sol#L14
- NetworkRestakeResetHook.sol#L14
- FullRestakeResetHook.sol#L14

The reset hook only resets the limits when a certain number of slashes occur within a period. This allows the operator to perform malicious actions without being penalized, as they know how many slashes they can make without consequences and within which period.

Recommendation

We recommend combining reset hooks with limits decrease in entity pairings.

Client's comments

The current contracts represent basic hook mechanics which can be used to implement more complex ones depending on demand in the future.

INFORMATIONAL-03	Codestyle issues	Fixed at: 1a873e1

Description

Lines:

- NetworkRestakeRedistributeHook.sol#L12 - double **IBaseSlasher** import.
- FullRestakeResetHook.sol#35, NetworkRestakeResetHook.sol#35, OperatorSpecificResetHook.sol#L34 - missing **PERIOD** zero check.

Also, consider using local variable naming for **_slashings** in the reset hooks contracts.

```
CircularBuffer.Bytes32CircularBuffer storage slashBuffer = _slashings[vault][operator];

if (slashBuffer.count() == 0) {
    slashBuffer.setup(SLASH_COUNT);
}
```

Recommendation

We recommend fixing codestyle issues.

Client's comments

It is intended that PERIOD can be 0.

INFORMATIONAL-04	Insufficient conditions when checking event for a slash in FullRestakeResetHook	Fixed at: 00e60be

Description

Line: FullRestakeResetHook.sol#L63

In the FullRestakeResetHook, if **operatorNetworkLimit** is zero, the function exits, and the slash is not counted. In **FullRestakeDelegator**, the operator's stake is determined based on three values: Vault's **activeStake**, **operatorNetworkLimit**, and **networkLimit**. Therefore, if any of these limits are zero, the operator's stake will also be zero. If the operator's stake is zero, they cannot participate in the network and thus cannot be slashed. Hence, when accounting for slashing in the reset hook, it is necessary to check not only **operatorNetworkLimit** but also **networkLimit** for zero values.

Recommendation

We recommend checking also if **networkLimit** equals zero in **FullRestakeResetHook** when returning from the function.

Client's comments

We reconsidered the behavior of all "reset" hooks so that even if the current delegations are zero, and the operator is slashed for the past actions, these slashings are also accounted for through the buffer.

INFORMATIONAL-05	Adding global buffer of subnetwork slashes in FullRestakeResetHook	Acknowledged

Description

Line: FullRestakeResetHook.sol#L14
In the FullRestakeResetHook, each vault–operator pairing for a specific subnetwork has its own buffer. In FullRestakeDelegator, setting the operator limit (**operatorNetworkLimit**) is not tied to the network limit (**networkLimit**), meaning they can be set independently. As a result, the total of all **operatorNetworkLimit** values for a specific subnetwork can exceed the **networkLimit**. Since these limits are set separately, adding a separate buffer for the vault–subnetwork pairing in the hook would provide more flexibility. This would allow limiting the total number of slashes for a specific subnetwork, which is currently not possible through the current implementation.

Recommendation

We recommended adding a global buffer for pairing vault-subnetwork to be able to account for the total slashing of certain subnetwork.

Client's comments

We may implement it depending on the demand.

STATE MIND