



Smart Contract Security Audit Report



Table Of Contents

1 Executive Summary	_____
2 Audit Methodology	_____
3 Project Overview	_____
3.1 Project Introduction	_____
3.2 Vulnerability Information	_____
4 Code Overview	_____
4.1 Contracts Description	_____
4.2 Visibility Description	_____
4.3 Vulnerability Summary	_____
5 Audit Result	_____
6 Statement	_____

1 Executive Summary

On 2026.08.17, the SlowMist security team received the RiftSwap Labs team's security audit application for hyperliquid-vault, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit
		Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This project is a modular on-chain asset management and cross-chain vault system. The system integrates both Hyperliquid accounts and the Topnod cross-chain router, supporting the transfer of local assets to remote management accounts, asynchronous settlement, and fund return. External applications mainly access the system via the deployed Vault proxy and its on-chain interfaces.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Missing Access Control Leading to Front-Running Tampering of Core Addresses	Authority Control Vulnerability Audit	Medium	Acknowledged
N2	Persistent Remote Position Causing Duplicate NAV Accounting	Design Logic Audit	Medium	Acknowledged
N3	Settlement Callback Lacks USDC Validation	Design Logic Audit	Suggestion	Acknowledged
N4	Token Compatibility Reminder	Others	Suggestion	Acknowledged
N5	Inconsistency Between API Wallet Disabled State and Asynchronous Execution Result of HyperCore	Design Logic Audit	Suggestion	Acknowledged
N6	Excessive Privilege Risk	Authority Control Vulnerability Audit	Medium	Acknowledged
N7	TopnodRouter Has Strong Trust Dependency on External TopnodProxy	Others	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

<https://github.com/r25it/curator2.0>

commit: f277c19e215bcbe01ae9c25b1dbed657e7d3b6e1

Audit Scope:

```
./src
├── adapters
│   ├── routers
│   ├── HyperAdapter.sol
│   └── TopnodAdapter.sol
```

<https://github.com/r25it/hyperliquid-vault>

commit: a1555085deae4b4e13e160159294a69cb89a1d78

Audit Scope:

```
./src
├── HyperliquidAccount.sol
├── TopnodEndpoint.sol
├── interfaces
│   ├── ICoreDepositWallet.sol
│   ├── ICoreWriter.sol
│   ├── IHyperliquidAccount.sol
│   ├── ITopnodEndpoint.sol
│   ├── ITopnodRouter.sol
│   └── ITopnodRouterCallBack.sol
├── libraries
│   ├── HyperCore.sol
│   └── Roles.sol
```

Fixed Version:

<https://github.com/r25it/curator2.0>

commit: 77d8208a6973984fa91e9e18100e932bc868fc5e

<https://github.com/r25it/hyperliquid-vault>

commit: d5d417af3f5f4257a6e79d714c15ad81871b6b29

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

TopnodEndpoint			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
bindHyperliquidAccount	External	Can Modify State	-
bindSourceAdapter	External	Can Modify State	-
onOrderSettled	External	Can Modify State	nonReentrant
bridgeToSourceChain	External	Can Modify State	onlyRole nonReentrant
__TopnodEndpoint_init	Internal	Can Modify State	onlyInitializing
__TopnodEndpoint_init_unchained	Internal	Can Modify State	onlyInitializing
_authorizeUpgrade	Internal	-	onlyRole
_consumeTrace	Private	Can Modify State	-

TopnodEndpoint			
_forwardSettledToken	Private	Can Modify State	-
_requireRouter	Private	-	-
_validateConfiguration	Private	-	-
_requireContract	Private	-	-

HyperliquidAccount			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
depositToPerp	External	Can Modify State	onlyRole nonReentrant
depositToSpot	External	Can Modify State	onlyRole nonReentrant
withdrawPerpToEvm	External	Can Modify State	onlyRole nonReentrant
withdrawSpotToEvm	External	Can Modify State	onlyRole nonReentrant
transferSpotToPerp	External	Can Modify State	onlyRole nonReentrant
transferPerpToSpot	External	Can Modify State	onlyRole nonReentrant
transferUsdcToEndpoint	External	Can Modify State	onlyRole nonReentrant
setApiWallet	External	Can Modify State	onlyRole nonReentrant
disableApiWallet	External	Can Modify State	onlyRole nonReentrant
emergencyDisableApiWallet	External	Can Modify State	onlyRole nonReentrant
emergencyCancelByOrderId	External	Can Modify State	onlyRole nonReentrant
emergencyCancelByCloid	External	Can Modify State	onlyRole nonReentrant

HyperliquidAccount			
usdc	External	-	-
coreWriter	External	-	-
coreDepositWallet	External	-	-
__HyperliquidAccount_init	Internal	Can Modify State	onlyInitializing
__HyperliquidAccount_init_unchained	Internal	Can Modify State	onlyInitializing
_authorizeUpgrade	Internal	-	onlyRole
_validateConfiguration	Private	-	-
_disableApiWallet	Private	Can Modify State	-

TopnodRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setAdapterAllowed	External	Can Modify State	onlyRole
setTrustedRouter	External	Can Modify State	onlyRole
depositERC20	External	Can Modify State	nonReentrant
onBatchOrderSettled	External	Can Modify State	nonReentrant
onOrderSettledV2	External	Can Modify State	nonReentrant
onOrderSettled	External	Can Modify State	nonReentrant
_forwardArraySettlement	Internal	Can Modify State	-

TopnodRouter			
__TopnodRouter_init	Internal	Can Modify State	onlyInitializing
__TopnodRouter_init_unchained	Internal	Can Modify State	onlyInitializing
_setTrustedRouter	Internal	Can Modify State	-
_authorizeUpgrade	Internal	-	onlyRole
_newTraceId	Internal	Can Modify State	-
_deriveTraceId	Internal	-	-
_consumeCallback	Internal	Can Modify State	-
_requireAllowedAdapter	Internal	-	-
_requireTopnodProxy	Internal	-	-
_validateRequest	Internal	-	-
_buildIntent	Internal	-	-
_encodeAggregatorPayload	Internal	-	-
_validateBusinessRequest	Internal	-	-
_receiverFor	Internal	-	-
_addressToBytes32	Internal	-	-
_forwardSettledTokens	Internal	Can Modify State	-

TopnodAdapter			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
TOPNOD_ROUTER	External	-	-

TopnodAdapter			
pendingOrderCount	External	-	-
pendingTraceIdAt	External	-	-
pendingOrderOf	External	-	-
execute	External	Can Modify State	onlyACModule
requiresEscrowInput	External	-	-
positions	External	-	-
onOrderSettled	External	Can Modify State	onlyTopnodCallback
__TopnodAdapter_init	Internal	Can Modify State	onlyInitializing
__TopnodAdapter_init_unchained	Internal	Can Modify State	onlyInitializing
_validatePlan	Internal	-	-
_decodeOrderData	Internal	-	-
_recordPendingOrder	Internal	Can Modify State	-
_requirePendingOrder	Internal	-	-
_removePendingOrder	Internal	Can Modify State	-
_validateStockSettlement	Internal	-	-

HyperAdapter			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyACModule

HyperAdapter			
requiresEscrowInput	External	-	-
positions	External	-	-
positionAssetDecimals	Public	-	-
onOrderSettled	External	Can Modify State	onlyTopnodRouter
__HyperAdapter_init	Internal	Can Modify State	onlyInitializing
__HyperAdapter_init_unchained	Internal	Can Modify State	onlyInitializing
_validatePlan	Internal	-	-
_decodeTransferData	Internal	-	-
_receiveReturn	Internal	Can Modify State	-
_localAsset	Internal	-	-
_bytes32ToAddress	Private	-	-

4.3 Vulnerability Summary

[N1] [Medium] Missing Access Control Leading to Front-Running Tampering of Core Addresses

Category: Authority Control Vulnerability Audit

Content

1. In the `bindHyperliquidAccount` function of the `TopnodEndpoint` contract, this function is responsible for setting the `hyperliquidAccount` that receives settlement funds in a single operation. The function imposes no restrictions on the caller. An attacker can front-run to bind a malicious contract after initialization. Existing validation only verifies the contract address and its returned `usdc()` and `bridgeEndpoint()` interfaces, which can be forged by malicious

contracts. After binding, `onOrderSettled` and `transferTokenToAccount` will continuously transfer tokens to this address, and the binding cannot be overwritten. This may result in theft of settlement funds.

2. In the `bindSourceAdapter` function of the `TopnodEndpoint` contract, the function is used to set the source-chain callback adapter. However, any address can front-run this function before legitimate configuration, and there is no validation for contract bytecode, routing relationships, or adapter whitelists. This address will subsequently be written as `callbackAdapter` for cross-chain requests. If the address is not whitelisted by the source-chain `TopnodRouter`, the callback may revert, leaving received funds stuck in the routing settlement flow. If bound to a `HyperAdapter` already permitted by another vault, tokens may be transferred to this adapter and recorded under the escrow of another vault.

Code Location:

`hyperliquid-vault/src/TopnodEndpoint.sol#L112-140`

```
function bindHyperliquidAccount(address account) external override {  
    ...  
}  
  
function bindSourceAdapter(address sourceAdapter_) external override {  
    ...  
}
```

Solution

Add strict access control to sensitive functions to verify the caller's identity, or complete all required configuration in one step during initialization.

Status

Acknowledged; The project team responded: The issue is known and will be addressed in the initialization function in the next version upgrade.

[N2] [Medium] Persistent Remote Position Causing Duplicate NAV Accounting

Category: Design Logic Audit

Content

In the positions function of the HyperAdapter contract, a `RemoteHeld` remote position with a fixed quantity of 1e6 is always returned. Meanwhile, `AssetSummedNAVOracle.totalAssets` counts both the escrow balance and adapter positions. When remote funds are cross-chain returned into escrow while off-chain prices remain within the fresh window and have not been updated, this fixed position is still included in valuation, resulting in duplicate valuation of the same funds. Early redeemers may receive assets based on an inflated NAV, with losses borne by remaining holders.

Code Location:

curator2.0/src/adapters/HyperAdapter.sol#L107-116

```
function positions() external view override(BaseAdapter, IAdapter) returns
(AdapterPosition[] memory positions_) {
    positions_ = new AdapterPosition[](1);
    positions_[0] = AdapterPosition({
        status: PositionStatus.RemoteHeld,
        chainId: destinationChainId,
        asset: remoteManagerAddress,
        amount: 10 ** uint256(REMOTE_POSITION_DECIMALS),
        refId: bytes32(0)
    });
}
```

Solution

It is recommended to record and update positions in the HyperAdapter contract based on the actual unsettled remote balance, and reduce the corresponding quantity upon refunds or successful settlements.

Status

Acknowledged; The project team responded: We update the price before each settlement.

[N3] [Suggestion] Settlement Callback Lacks USDC Validation

Category: Design Logic Audit

Content

In the `onOrderSettled` function of the `TopnodEndpoint` contract, only the lengths of the `tokens` and `amounts` arrays are checked; there is no enforcement that `tokens[0] == usdc`. Malicious proxies, erroneous callbacks, or operational misconfiguration can cause non-USDC tokens to be transferred into `HyperliquidAccount`. The outbound transfer path for this account hardcodes HyperCore USDC, and there is no generic token withdrawal mechanism. As a result, non-USDC assets may be permanently locked.

Code Location:

`hyperliquid-vault/src/TopnodEndpoint.sol#L143-163`

```
function onOrderSettled(
    bytes32 traceId,
    bool,
    address[] calldata tokens,
    uint256[] calldata amounts,
    bytes calldata
) external override nonReentrant {
    ...

    address token = tokens[0];
    uint256 amount = amounts[0];

    ...

    _forwardTokenToAccount(token, amount, hyperliquidAccount);
}
```

Solution

Enforce validation that the settlement token is the configured usdc before consuming the trace; reject zero-amount transfers and unsupported assets.

Status

Acknowledged; The project team responded: It will not affect the system, and we are not considering fixing it at the moment.

[N4] [Suggestion] Token Compatibility Reminder

Category: Others

Content

In the depositERC20 function of the TopnodRouter contract, after calling safeTransferFrom, the contract directly authorizes and submits the requested inputAmount to TopnodProxy without comparing the router's balance before and after the transfer. For tax-deducting or deflationary ERC-20 tokens, the actual received amount may be less than inputAmount. This creates a mismatch between the submitted amount and real holdings, which can lead to order reverts, funds being stuck within the Router, or accounting discrepancies in cross-chain operations.

Code Location:

curator2.0/src/adapters/routers/TopnodRouter.sol#L121-138

```
function depositERC20(ITopnodRouter.DepositRequest calldata request)
    external
    override
    nonReentrant
    returns (bytes32 traceId)
{
    ...

    IERC20(request.inputToken).safeTransferFrom(msg.sender, address(this),
request.inputAmount);
    IERC20(request.inputToken).forceApprove(topnodProxy, request.inputAmount);

    ...
}
```

Solution

Record balances before and after the transfer to calculate the actual received amount. If the protocol requires an exact amount, the received value must equal the requested value; otherwise, proceed with submission using the

actual received amount, and update events and downstream parameters accordingly.

Status

Acknowledged; The project team responded: We will not support deflationary or other special tokens.

[N5] [Suggestion] Inconsistency Between API Wallet Disabled State and Asynchronous Execution

Result of HyperCore

Category: Design Logic Audit

Content

In the `_disableApiWallet` function of the `HyperliquidAccount` contract, both `disableApiWallet` and `emergencyDisableApiWallet` share this logic. The function first updates the EVM-side `apiWallet` to a deterministic fallback address, then submits a disabled request to `CoreWriter` via `HyperCore.setApiWallet`. `CoreWriter` only indicates that the request has been submitted and cannot prove that `HyperCore` has completed execution. If subsequent execution fails, the contract state will incorrectly show that the old API Wallet has been replaced, while the old wallet may still retain transaction permissions in reality.

`setApiWallet` follows the same pattern: updating local state first, then submitting an asynchronous request.

Code Location:

`hyperliquid-vault/src/HyperliquidAccount.sol`

```
function setApiWallet(address newApiWallet, string calldata newApiWalletName)
    external
    override
    onlyRole(API_WALLET_MANAGER_ROLE)
    nonReentrant
{
    ...

    apiWallet = newApiWallet;
    isApiWalletUsed[newApiWallet] = true;
    HyperCore.setApiWallet(newApiWallet, newApiWalletName);

    emit ApiWalletSet(previousApiWallet, newApiWallet, newApiWalletName);
}
```

```
function _disableApiWallet(bool emergency) private {
    ...

    address replacementApiWallet = HyperCore.disabledApiWallet(name);

    apiWallet = replacementApiWallet;
    isApiWalletUsed[replacementApiWallet] = true;
    HyperCore.setApiWallet(replacementApiWallet, name);

    ...
}
```

Solution

Adopt a request-pending and confirmation mechanism. Upon submitting an operation, only record the pending wallet and request status. A trusted off-chain service shall call a restricted confirmation function after verifying successful HyperCore execution to update `apiWallet`. During request failure, timeout, or unconfirmed periods, preserve the original state and provide cancellation or retry workflows.

Status

Acknowledged; The project team responded that if hypercore ultimately fails, the off-chain request will be re-initiated.

[N6] [Medium] Excessive Privilege Risk

Category: Authority Control Vulnerability Audit

Content

1. In the `_authorizeUpgrade` function of the `TopnodEndpoint`, `HyperliquidAccount` and `TopnodRouter` contracts, the `UPGRADER_ROLE` or `DEFAULT_ADMIN_ROLE` can directly authorize upgrades to any UUPS implementation. If the private key for these roles is lost or compromised, the implementation contract could be upgraded to a malicious contract, ultimately endangering fund security within the contracts.

Code Location:

hyperliquid-vault/src/TopnodEndpoint.sol#L249

hyperliquid-vault/src/HyperliquidAccount.sol#L209

```
function _authorizeUpgrade(address) internal view override onlyRole(UPGRADER_ROLE)
{}
```

curator2.0/src/adapters/routers/TopnodRouter.sol#L247

```
function _authorizeUpgrade(address) internal view override
onlyRole(DEFAULT_ADMIN_ROLE) {}
```

2. In the `setAdapterAllowed` function of the `TopnodRouter` contract, the `ADAPTER_MANAGER_ROLE` is permitted to add any non-zero address to the whitelist, without validating contract bytecode, adapter identity, supported assets, or cross-chain routing logic. Once whitelisted, the address is allowed to call `depositERC20` and participate in the Router's callback and fund-forwarding workflows.

Code Location:

curator2.0/src/adapters/routers/TopnodRouter.sol#L108-113

```
function setAdapterAllowed(address adapter, bool allowed) external
onlyRole(ADAPTER_MANAGER_ROLE) {
    ...
}
```

3. In the `setTrustedRouter` function of the `TopnodRouter` contract, the `OPERATOR_ROLE` can set a receiving router for any target chain, with only a non-zero address check performed. `_receiverFor` directly uses this address to construct cross-chain intents. In the event of compromised operational privileges or misconfiguration, subsequent cross-chain funds may be sent to malicious routers, incorrect contracts, or unusable addresses.

Code Location:

curator2.0/src/adapters/routers/TopnodRouter.sol#L116-118

```
function setTrustedRouter(uint256 destinationChainId, address router) external
onlyRole(OPERATOR_ROLE) {
```

```
_setTrustedRouter(destinationChainId, router);  
}
```

Solution

In the short term, transferring the ownership of core roles to time-locked contracts and managing them by multi-signatures is an effective solution to avoid single-point risks. However, in the long run, a more reasonable solution is to implement a permission separation strategy and set up multiple privileged roles to manage each privileged function separately. Permissions involving user funds and contract core parameter updates should be managed by the community, while permissions involving emergency contract suspensions can be managed by EOA addresses. This ensures the safety of user funds while responding quickly to threats.

Status

Acknowledged; The project team responded that they will use multi-signature and KMS management permissions.

[N7] [Information] TopnodRouter Has Strong Trust Dependency on External TopnodProxy

Category: Others

Content

The deposit flow of TopnodRouter approves tokens to the configured topnodProxy and invokes its methods. The settlement flow also fully trusts callbacks originating from this address. If TopnodProxy contains vulnerabilities or is maliciously upgraded, it may leverage Router's approvals to transfer away tokens, or forge callback data. This can cause the Router to forward its held assets to whitelisted adapters, or lock funds inside the external TopnodProxy.

Code Location:

curator2.0/src/adapters/routers/TopnodRouter.sol

```
function depositERC20(ITopnodRouter.DepositRequest calldata request)  
    external  
    override  
    nonReentrant  
    returns (bytes32 traceId)  
{  
    ...  
}
```

```

        ITopnodProxy(topnodProxy).depositERC20(request.inputToken,
request.inputAmount, traceId, routedIntent);

        ...
    }

    function onBatchOrderSettled(
        bytes32 traceId,
        bool success,
        address[] calldata tokens,
        uint256[] calldata amounts,
        bytes calldata data
    ) external override nonReentrant {
        _requireTopnodProxy();
        _forwardArraySettlement(traceId, success, tokens, amounts, data);
    }

    function onOrderSettledV2(
        bytes32 traceId,
        bool success,
        address[] calldata tokens,
        uint256[] calldata amounts,
        bytes calldata data
    ) external override nonReentrant {
        _requireTopnodProxy();
        _forwardArraySettlement(traceId, success, tokens, amounts, data);
    }

    function onOrderSettled(bytes32 traceId, bool success, address token, uint256
amount, bytes calldata data)
        external
        override
        nonReentrant
    {
        _requireTopnodProxy();

        ...
    }

```

Solution

N/A

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002608200003	SlowMist Security Team	2026.08.17 - 2026.08.20	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 3 medium risk vulnerabilities, 3 suggestions and 1 information. All the findings were acknowledged. Since the permission of the core role has not yet been transferred, the risk level reported is temporarily medium.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.



Official Website
www.slowmist.com



E-mail
team@slowmist.com



Twitter
[@SlowMist_Team](https://twitter.com/SlowMist_Team)



Github
<https://github.com/slowmist>