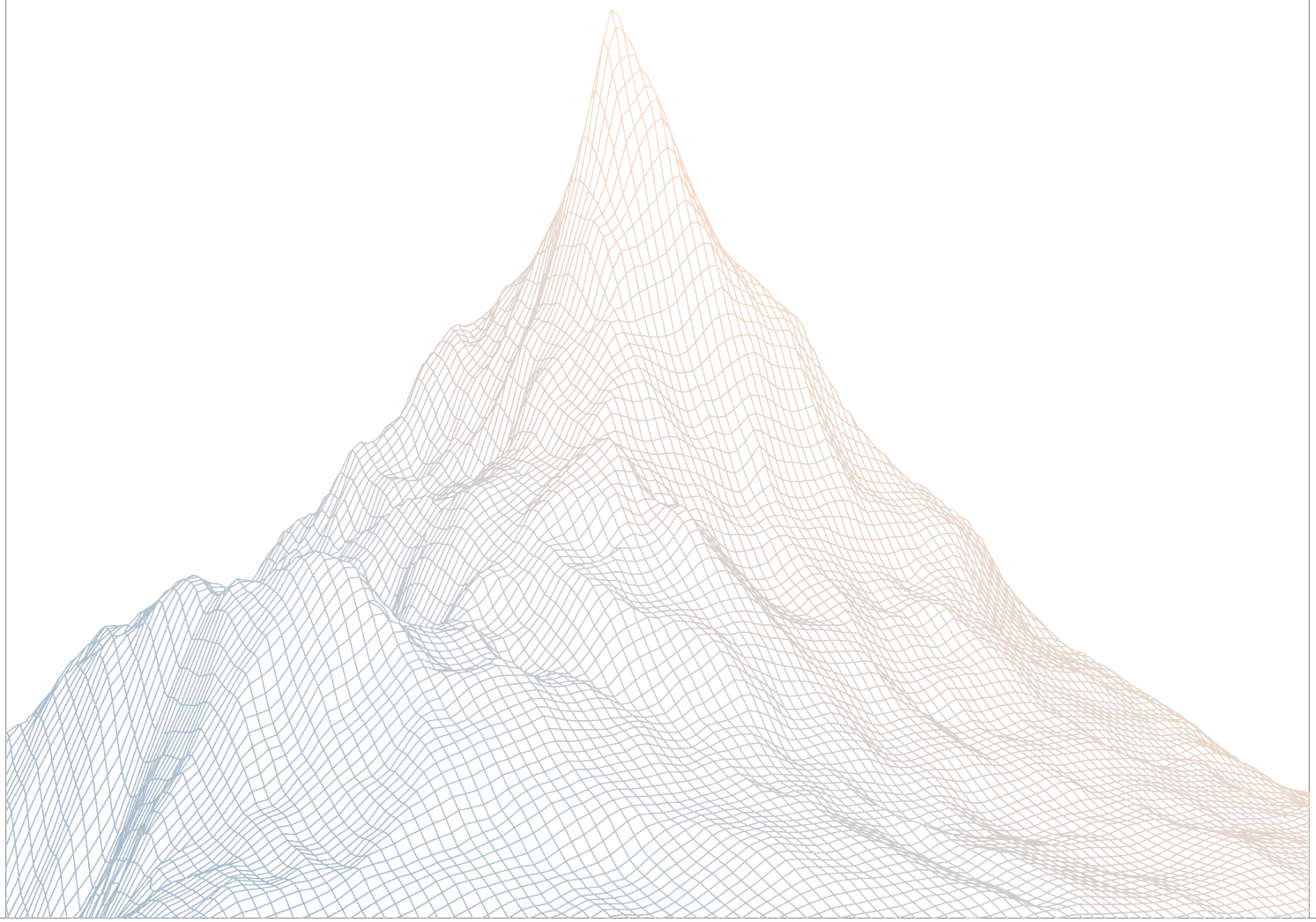
Zenith

Hyperwave

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: August 12th to August 21th, 2025

AUDITED BY: adriro
said

Contents

1 Introduction 2

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary 3

2.1 About Hyperwave 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary 5

4 Findings 6

4.1 Low Risk 7

4.2 Informational 10

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Hyperwave

The first ever liquid token that accrues yield from the Hyperliquidity Provider vault (HLP).

hwHLP lets you profit from market making and liquidations on Hypercore, while retaining your liquid assets for DeFi on HyperEVM and beyond.

2.2 Scope

The engagement involved a review of the following targets:

Target	hlp-corewriter
Repository	https://github.com/SwellNetwork/hlp-corewriter
Commit Hash	`5a19bb4373eaf3eb57d872f81d20177fb5cf9b2b`

Files	`util/*`
	`Configurator.sol`
	`HyperCoreAccount.sol`
	`TradeStakeManager.sol`
	`VaultManager.sol`

2.3 Audit Timeline

August 12, 2025	Audit start
August 21, 2025	Audit end
August 22, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	8
Total Issues	**10**

3

Findings Summary

ID	Description	Status
L-1	Missing token validation in withdraw()	Resolved
L-2	USDC token cannot be registered	Acknowledged
I-1	Simplify bridge address derivation	Resolved
I-2	USDC transfers cannot be disabled	Resolved
I-3	vaultWithdraw does not check the vault's lock time.	Acknowledged
I-4	VaultManager lacks an account equity getter for the deposited vault	Acknowledged
I-5	Unused constant in RolesLib	Resolved
I-6	Indexed fields on HyperCoreAccount events	Resolved
I-7	Support for non-USDC markets	Resolved
I-8	Whitelist checks on several exit operations can cause issue	Resolved

4

Findings

4.1 Low Risk

A total of 2 low risk findings were identified.

[L-1] Missing token validation in `withdraw()`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- TradeStakeManager.sol

Description:

The implementation of `withdraw()` in TradeStakeManager doesn't check if the token is actually enabled in the Configurator.

Recommendations:

Consider adding a validation to ensure the given `tokenAddress` is a registered token in the Configurator contract, as `withdrawNative()` does.

Swell: We think that the inverse needs to be true here, the check in `withdrawNative()` should be removed. Addressed in PR 44.

Zenith: Verified. Both `withdraw()` and `withdrawNative()` only check the whitelist, without checking if the token is configured.

[L-2] USDC token cannot be registered

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Configurator.sol

Description:

The Configurator contract determines if a token is registered by checking if the corresponding `tokenIndex` is zero. A zero token index means the token is not enabled, which leads to the implementation of `enableToken()` to correctly disallow a zero `tokenIndex`.

```
45:     function enableToken(uint32 tokenIndex)
    external onlyRole(RolesLib.GOVERNOR) {
46:         if (tokenIndex == 0) revert Zero("tokenIndex");
```

This behavior causes the edge case of not being able to enable USDC, the token placed at the zero index. While the implementation handles this case in particular, for example by having a dedicated `spotSendUSDC()` function, there are other places that could be affected if USDC eventually gets an EVM representation (already available on testnet).

One such case is the `forward()` function that executes an EVM → Core bridge. Given that the implementation explicitly checks for `tokenIndex ≠ 0`, this would prevent USDC from being sent to the L1 layer.

```
461:         ISupportedTokens.Token memory token = _getToken(tokenAddress);
462:         address bridgeAddress = token.bridgeAddress;
463:         if (token.tokenIndex == 0 || bridgeAddress == address(0)) {
464:             revert TokenNotConfigured(tokenAddress);
465:         }
```

Recommendations:

Add a dedicated `enabled` flag in the configurator instead of relying on `tokenIndex ≠ 0`.

Swell: We believe that USDC may have a different implementation on HyperEVM, potentially in the way the bridgingAddress is computed. Since USDC has not yet been deployed, there may be a bespoke implementation for this asset once it goes live.

Zenith: Acknowledged.

4.2 Informational

A total of 8 informational findings were identified.

[I-1] Simplify bridge address derivation

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- Configurator.sol

Description:

The implementation of `enableToken()` applies multiple manipulations over the bytes domain to derive the bridge address, which can be simplified by appending the index to the base system address.

```
65:           // Compute the expected bridge address based on the token index
66:            // Move the token index 12 bytes to the left so as to prevent
    truncating on cast to bytes20
67:            bytes20 formattedTokenIndex
    = bytes20(bytes32(uint256(tokenIndex)) << 8 * 12);
68:            // Add the 2 prefix for all bridges and convert bytes20 to
    address
69:            bridgeAddress = address(formattedTokenIndex |
    hex"2000000000000000000000000000000000000000");
```

Recommendations:

```
bridgeAddress = address(uint160(0x2000000000000000000000000000000000000000)
    | tokenIndex)
```

Swell: Addressed in PR-44.

Zenith: Verified.

[I-2] USDC transfers cannot be disabled

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- TradeStakeManager.sol
- VaultManager.sol

Description:

While the `spotSend()` function, used to transfer arbitrary ERC20 tokens, performs a check against the Configurator contract to validate that the token is enabled, its matching `spotSendUSDC()` variant always succeeds as long as the destination is whitelisted.

Recommendations:

Ensure this behavior is correct, as this would also allow the operator to freely transfer USDC.

Swell: Resolved with PR-44.

Zenith: Verified. USDC can now be controlled using a spot transfer whitelist, while withdrawals have a separate whitelist.

[I-3] `vaultWithdraw` does not check the vault's lock time.

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Acknowledged	`LIKELIHOOD:` Low

Target

- VaultManager.sol#L154-L171

Description:

When `vaultWithdraw` is called, it attempts to withdraw an account's share from the specified vault. However, it does not verify whether the lock-up period has passed. The HLP has a 4-day lock-up period, while user vaults have a 1-day lock-up period.

Consider adding a check for the lock-up period to provide a more descriptive revert message if the withdrawal fails due to the lock-up restriction.

Recommendations:

Swell: We acknowledge and choose not to implement this in the contract. If invalid, HyperCore will reject the request. There are other similar validations we could do that fit into the same category, that we are omitting to keep code size manageable.

[I-4] `VaultManager` lacks an account equity getter for the deposited vault

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- VaultManager.sol

Description:

Inside `VaultManager`, there is no way to retrieve information about an account's equity in the deposited vault. Such a getter would be helpful for monitoring PnL and making decisions on whether to stay in or withdraw from the vault.

```
function getAccountVaultEquity(address account, address vault)
    public
    view
    returns (uint64 equity, uint64 lockedUntilTimestamp)
{

    L1ReadLib.UserVaultEquity memory vaultEquity = L1ReadLib.getUserVaultEquit
        y(account, vault);
    return (vaultEquity.equity, vaultEquity.lockedUntilTimestamp);
}
```

Recommendations:

Swell: We acknowledge and choose not to implement this in the contract. We will source this information through a deployment of L1Read, and use it as a data source in the offchain component.

[I-5] Unused constant in RolesLib

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- RolesLib.sol

Description:

The `DEFAULT_ADMIN_ROLE` constant defined in RolesLib has the same value as the constant of the same name in the OpenZeppelin library, but contracts actually refer to the OpenZeppelin variant, rather than the one in RolesLib.

```
37:         _grantRole(DEFAULT_ADMIN_ROLE, governor);
```

Recommendations:

Take into account that changing its value in RolesLib would cause a desync with respect to the actual value being used in the contracts, and consider deleting it to prevent potential accidents.

Swell: Resolved with PR-44.

Zenith: Verified. Fixed by removing the constant.

[I-6] Indexed fields on HyperCoreAccount events

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- IHyperCoreAccount.sol

Description:

Unlike other contracts, none of the events emitted from the HyperCoreAccount have indexed fields.

Recommendations:

Consider switching to indexed fields for `address` parameters and indices, such as the asset index or the order ID.

Swell: Resolved with PR-44.

Zenith: Verified. Fixed as recommended.

[I-7] Support for non-USDC markets

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- Configurator.sol

Description:

Already available on testnet, Hyperliquid is expanding to support spot markets where the quote token may differ from USDC.

This means that, eventually, there might be multiple markets for the same base token, affecting the assumptions in the implementation of `_getTokenIndexForSpotIndex()`:

```
249:     /**
250:      * @dev Gets the primary token index for a spot index from L1Read
    (USDC markets are [<tokenIndex>, 0])
251:      * @dev Reverts if spot index is not found in Hyperliquid
252:      * @param spotIndex The zero-based spot index to query
253:      * @return The token index for the primary token in the spot market
254:      */
255:     function _getTokenIndexForSpotIndex(uint32 spotIndex)
    internal view returns (uint64) {
256:         L1ReadLib.SpotInfo memory spotInfo
    = L1ReadLib.getSpotInfo(spotIndex);
257:         return spotInfo.tokens[0]; // The first token index is the
    primary one
258:     }
```

Recommendations:

Ensure the spot market is the correct one by validating that the second element of the `spotInfo.tokens` array is zero.

```
function _getTokenIndexForSpotIndex(uint32 spotIndex)
    internal view returns (uint64) {
```

```
    L1ReadLib.SpotInfo memory spotInfo = L1ReadLib.getSpotInfo(spotIndex);
  if (spotInfo.tokens[1] != 0) revert NonUsdcMarket();
    return spotInfo.tokens[0]; // The first token index is the primary one
}
```

Swell: Resolved with PR-44.

Zenith: Verified.

[I-8] Whitelist checks on several exit operations can cause issue

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- VaultManager.sol#L162-L164
- TradeStakeManager.sol#L200-L202
- TradeStakeManager.sol#L293-L295
- TradeStakeManager.sol#L333-L335

Description:

When governance de-lists an entity (validator/perp/vault/token) by removing its whitelist, operators may still need to cancel existing exposure (`undelegate`, `cancelSpotOrderByOid`, `cancelPerpOrderByOid` and `vaultWithdraw`). Current checks block these "exit" operations if the target is no longer whitelisted/supported, potentially stranding positions until governance re-whitelists.

Recommendations:

Allow exit operations without a per-target whitelist, as these operations are safe.

Swell: Resolved with PR-44.

Zenith: Verified.