CODESPECT

HyperLiquid Forwarder

SECURITY ASSESSMENT REPORT

June 9, 2025

Prepared for

Swell

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Tailored Cybersecurity Solutions: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

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

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

4 Executive Summary

This document presents the security assessment conducted by CODESPECT for the smart contracts of Swell. Swell is a non-custodial staking protocol offering liquid staking and restaking tokens.

This audit focuses on a newly developed contract by the Swell team, designed to forward USDT0 and USDe tokens from the Nucleus Boring Vault on HyperEVM to the HyperCore system.

The audit was performed using:

a) Manual analysis of the codebase.

CODESPECT found three points of attention, one classified as `High` and two classified as `Informational`. All of the issues are summarised in Table 2.

Organization of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the update of the Repricing oracle.
- **Section 7** presents the issues.
- **Section 7** discusses the documentation provided by the client for this audit.
- **Section 8** presents the compilation and tests.

Severity	Unresolved	Fixed	Acknowledged
High	0	1	0
Informational	0	1	1
Total	**0**	**2**	**1**

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	Swell
Type of Project	Forwarder
Duration of Engagement	1 Day
Duration of Fix Review Phase	1 Day
Draft Report	June 8, 2025
Final Report	June 9, 2025
Repository	hyperliquid-forwarder
Commit (Audit)	59910084b7669f721330be6dc68557b6ac747c4b
Commit (Final)	56e4679ae55aff64738274209fb526774f3a2d54
Documentation Assessment	High
Test Suite Assessment	High
Auditors	Talfao, Kalogerone

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	LoC
1	HyperliquidForwarder.sol	38
	Total	**38**

5.2 Findings Overview

	Finding	Severity	Update
1	The `HYPE` bridge address is hardcoded as the `WHYPE` bridge address	High	Fixed
2	The arbitrary `evmEOAToSendToAndForwardToL1` parameter allows for sending tokens to the wrong address	Info	Fixed
3	There are no checks that the system address on HyperCore has sufficient supply	Info	Acknowledged

6 System Overview

The `HyperLiquidForwarder` contract acts as a bridge facilitator for transferring ERC20 tokens from HyperEVM (Layer 2) to Hyperliquid's HyperCore (Layer 1). It ensures the correct routing of tokens through Hyperliquid's bridge infrastructure.

The contract maintains a mapping of token addresses to their respective bridge addresses. This mapping can be configured by the contract owner via the `addTokenIDToBridgeMapping(...)` function:

```
function addTokenIDToBridgeMapping(
    address tokenAddress,
    address bridgeAddress,
    uint16 tokenID
) public requiresAuth;
```

In the Hyperliquid ecosystem, each ERC20 token available on HyperCore (L1) has a corresponding bridge address. The bridge address follows a deterministic format defined by the system:

Each token has a system address on the Core, which consists of the byte `0x20` *as the first byte, followed by 19 zero bytes, with the final 2 bytes representing the token index in big-endian format. For instance, token index 200 would have the system address* `0x2000000000000000000000000000000000000c8`. [docs]

The Swell team is responsible for accurately configuring token and bridge addresses, and for thoroughly testing the configuration before initiating any asset transfers from the Nucleus vault.

To forward tokens from the vault to HyperCore, authorized managers of vault will use the `forward(...)` function:

```
function forward(
    ERC20 token,
    uint256 amount,
    address evmEOAToSendToAndForwardToL1
) external;
```

This function performs two sequential operations:

a. Transfers the specified `amount` of `token` from the vault to an externally owned address (EOA) managed by the Swell team.

b. Immediately forwards the tokens from that EOA to the corresponding bridge address.

Once the tokens reach the bridge address, the Hyperliquid system reflects the transfer in the balance of the EOA on HyperCore (L1), enabling the Swell team to manage the assets from there.

7 Issues

7.1 [High] The HYPE bridge address is hardcoded as the WHYPE bridge address

File(s): HyperliquidForwarder.sol

Description: The HyperliquidForwarder contract has 2 hardcoded addresses:

```
address public constant WHYPE = 0x5555555555555555555555555555555555555555;
address private constant WHYPE_BRIDGE = 0x2222222222222222222222222222222222222222;
```

However, according to the HyperLiquid docs (link), the 0x2222222222222222222222222222222222222222 address is HYPE's system address and not WHYPE's. These values are implemented in the addTokenIDToBridgeMapping(...) function:

```
function addTokenIDToBridgeMapping(address tokenAddress, address bridgeAddress, uint16 tokenID)
    public
    requiresAuth
{
    // HYPE/WHYPE is an exception and is handled separately, do not allow an owner to incorrectly set it
    if (tokenAddress == WHYPE) {
        tokenAddressToBridge[WHYPE] = WHYPE_BRIDGE;
        return;
    }
    ...
}
```

Impact: Any WHYPE tokens that will be sent through this contract will be lost.

Recommendation(s): Replace the WHYPE for the HYPE token to correctly use the bridge. However, HYPE transfer should be treated like native ETH transfer; therefore, the implementation of the forward(...) function should be changed.

Status: Fixed

Update from Swell: Resolved with commit 56e4679ae55aff64738274209fb526774f3a2d54

7.2 [Info] The arbitrary evmEOAToSendToAndForwardToL1 parameter allows for sending tokens to the wrong address

File(s): HyperliquidForwarder.sol

Description: In the forward(...) function, the msg.sender sends his own tokens from the HyperEVM to the HyperCore where the evmEOAToSendToAndForwardToL1 address receives them. Considering that the protocol will use 5 multi-sig addresses, such an address could be verified against them in contract level to remove the possibility of sending tokens to the wrong address.

Status: Fixed

Update from Swell: Resolved with commit 56e4679ae55aff64738274209fb526774f3a2d54

7.3 [Info] There are no checks that the system address on HyperCore has sufficient supply

File(s): HyperliquidForwarder.sol

Description: According to the HyperLiquid documentation, when bridging tokens between HyperEVM and HyperCore, there are currently no checks to verify that the system address has sufficient supply. Furthermore, these checks cannot be implemented at the contract level, meaning transactions that send funds to the bridge cannot be reverted automatically in such cases.

Therefore, the protocol team should always exercise caution before performing large token transfers.

Status: Acknowledged

Update from Swell: Acknowledged

8 Evaluation of Provided Documentation

The Swell documentation was provided in two forms:

- **Natspec Comments**: The smart contract included well-written Natspec comments, which significantly aided in understanding the functionality and purpose of each component within the protocol.
- **Internal Documentation**: The Swell team also shared internal specifications and flow diagrams outlining the roles and interactions of the contract. This provided a clear high-level overview of the system architecture and design intentions.

Throughout the audit process, the Swell team remained responsive and cooperative, promptly addressing all questions raised by the CODESPECT team. This greatly contributed to the smooth and efficient execution of the audit.

9 Test Suite Evaluation

9.1 Compilation Output

```
> forge compile
[] Compiling...
[] Compiling 25 files with Solc 0.8.21
[] Solc 0.8.21 finished in 1.74s
Compiler run successful!
```

9.2 Tests Output

```
> forge test
Ran 7 tests for test/HyperliquidForwarder.t.sol:HyperliquidForwarderTest
[PASS] test_AddTokenIDToBridgeMapping_Reverts_When_BridgeMismatch() (gas: 21139)
[PASS] test_AddTokenIDToBridgeMapping_Reverts_When_NotOwner() (gas: 23213)
[PASS] test_Forward_Reverts_If_BridgeNotSet() (gas: 97831)
[PASS] test_Forward_Reverts_If_UserHasNoAllowance() (gas: 105207)
[PASS] test_Forward_Reverts_If_evmEOAHasNoAllowance() (gas: 122938)
[PASS] test_Forward_Succeeds_TwoStepTransfer() (gas: 175407)
[PASS] test_Forward_Succeeds_USDT0() (gas: 368600)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 8.67s (4.11s CPU time)
```

9.3 Notes on the Test Suite

The Swell team developed a test suite that is appropriate for the size and complexity of the contract. It covers a range of both successful and failure scenarios, effectively validating the core logic of the contract under typical conditions.

However, due to the inherent limitations of testing within Foundry—particularly its inability to fully emulate cross-system bridging—the suite could not detect issues related to incorrect bridge-token pair configurations. As a result, a vulnerability involving mismatched bridge and token addresses persisted in the contract.

We therefore strongly recommend conducting production testing with minimal token amounts to verify that the contract behaves as intended in a live environment.