D2 Hype CoreWriter Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Immeas

MrPotatoMagic

October 16, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

D2 Hype Module is a new addition to the D2 strategies. It allows the D2 trader to transfer tokens back and forth between HyperEVM and HyperCore. On HyperCore, the trader can execute trades and then transfer the results back to the D2 vaults and their stakers.

4.1 Actors and Roles

- **1. Actors:**
  - **Trader:** Executes trades using the D2 strategies.
- **2. Roles:**
  - `EXECUTOR_ROLE`: Conducts all trading activities across different modules.

4.2 Key Components

- `Hype_Module`: Facilitates the calls between HyperEVM and HyperCore. All calls are limited to the `EXECUTOR_ROLE`. Note that when transferring tokens between HyperCore and HyperEVM, some tokens use different decimals on each platform. This must be considered by the `EXECUTOR_ROLE` to avoid unintentionally burning dust.

4.3 Centralization Risks

The `EXECUTOR_ROLE`, a protocol-controlled wallet, is of particular importance as it is responsible for active trading. This reliance introduces potential risks, including the possibility of private key leaks.

5 Audit Scope

```
contracts/modules/Hype.sol
```

6 Executive Summary

Over the course of 2 days, the Cyfrin team conducted an audit on the D2 Hype CoreWriter smart contracts provided by D2. In this period, a total of 7 issues were found.

During the audit, two low-severity issues were identified, first one for missing support for transferring the native gas token (HYPE) to HyperCore via a native value transfer. The second one for use of raw ERC20 `transfer` instead of best-practice `SafeERC20` methods.

Two informational items were also reported: the `nonReentrant` modifier should appear first in the modifier list, “magic numbers” used where named constants could be more descriptive and a parameter name mismatch. Additionally, two minor gas optimizations were noted.

Summary

Project Name	D2 Hype CoreWriter
Repository	d2-contracts
Commit	b7844b60ec6b...
Fix Commit	134a2b1c4d40...
Audit Timeline	Oct 13th - Oct 14th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	3
Gas Optimizations	2
Total Issues	7

Summary of Findings

[L-1] Native HYPE transfers to HyperCore will not work	Resolved
[L-2] Unsafe ERC20 transfers	Acknowledged
[I-1] `nonReentrant` not the first modifier	Resolved

[I-2] Consider using constants instead of magic numbers for action IDs	Resolved
[I-3] Parameter name mismatch between interface and implementation may be misleading	Resolved
[G-1] Remove loop when sending actions	Resolved
[G-2] `Hype_module::assetAddress` can be `pure`	Resolved

7 Findings

7.1 Low Risk

7.1.1 Native HYPE transfers to HyperCore will not work

Description: `Hype_Module::hyper_depositSpot` always calls `IERC20(token).transfer(assetAddress(uint64(asset)), _wei)`, including when `asset == 150` (native HYPE). Per HyperCore docs, HYPE (id `150`) must be transferred as native value to the special system address `0x2222...2222`, not via ERC-20. In addition, when transferring native HYPE the `token` parameter should be a placeholder (the zero address) to avoid ambiguity and accidental ERC-20 paths.

Impact: Native HYPE transfers will likely not work as the token parameter will either be a placeholder or, worst case a wrapped version where the transfer might succeed but the crediting of tokens on HyperCore will fail (as the asset id is wrong) resulting in lost tokens. It can also create ambiguity around the `token` parameter for native deposits can lead to misconfiguration or silent misrouting of funds.

Recommended mitigation:

- Make `hyper_depositSpot payable` and branch on `asset`:
  - If `asset == 150` (HYPE/native): require `token == address(0)` and `msg.value == _wei`, then send native value to `0x2222...2222`:

```
    function hyper_depositSpot(
        address token,
        uint32 asset,
        uint64 _wei
-   ) external onlyRole(EXECUTOR_ROLE) nonReentrant {
+   ) external payable onlyRole(EXECUTOR_ROLE) nonReentrant {
+       if (asset == 150) {
+           require(token == address(0), "token must be zero for HYPE");
+           require(msg.value == _wei, "msg.value mismatch");
+           (bool ok, ) = address(0x2222222222222222222222222222222222222222).call{value:
↪   _wei}("");
+           require(ok, "native HYPE transfer failed");
+           return;
+       }

        IERC20(token).transfer(assetAddress(uint64(asset)), _wei);
    }
```

- Optionally overload or split the API (`hyper_depositSpotHype(uint64 _wei)` vs `hyper_depositSpotERC20(address token, uint32 asset, uint256 amount)`) to remove ambiguity.

D2: Fixed in commit 134a2b1 except for the part where we make it payable and check the msg.value as it's meant to transfer funds IN the strategy contract, not funds owned by the sender / operator.

Cyfrin: Verified. If `asset == 150` the Hype module now calls `0x22...22` with `value: _wei`.

7.1.2 Unsafe ERC20 transfers

Description: `Hype_Module::hyper_depositSpot` uses `IERC20(token).transfer(...)` directly and ignores the return value. Many ERC-20s are non-standard (e.g., USDT) and either don't return `bool` or revert on failure in non-obvious ways, making bare `transfer`/`transferFrom` unsafe.

Impact: Token transfers can silently fail or behave inconsistently across tokens, causing deposits not to be credited on Core and potentially leaving funds stranded in the caller.

Recommended mitigation: Use OpenZeppelin's `SafeERC20` for all token interactions:

```
using SafeERC20 for IERC20;
```

```
IERC20(token).safeTransfer(assetAddress(uint64(asset)), amount);
```

D2: Acknowledged. We're not on mainnet and all tokens most likely use proper compliant ERC20 implementations, so we're skipping on adding SafeERC20 in. We will vet tokens we support in vaults before whitelisting and using.

7.2 Informational

7.2.1 `nonReentrant` not the first modifier

Description Across `Hype_Module`, `nonReentrant` is listed after `onlyRole(EXECUTOR_ROLE)` on external functions. In Solidity, modifiers are applied left-to-right, and the first modifier becomes the outermost wrapper. For consistent defense-in-depth, `nonReentrant` should be first so it also guards any logic within subsequent modifiers (present or future), minimizing the risk that a modifier could perform state changes or external calls before the reentrancy guard is set.

Consider changing the functions to have `nonReentrant` as the first modifier.

D2: Fixed in commit `c5aeb40`

Cyfrin: Verified.

7.2.2 Consider using constants instead of magic numbers for action IDs

Description: The action IDs that are used to interact with HyperCore are currently hardcoded in Hype.sol module's functions.

For example, action ID = 6 represents the `spotSend` action:

```
function hyper_sendSpot(
        uint64 asset,
        uint64 _wei
    ) external onlyRole(EXECUTOR_ROLE) nonReentrant {
        sendAction(6, abi.encode(assetAddress(asset), asset, _wei));
    }
```

Impact: While this poses no risk, using constants instead of magic numbers improve code maintainability and readability to explain the magic number's intended purpose.

Recommended Mitigation: Consider implementing constants for each action ID hardcoded currently. For example, an action ID of 1 can be named as constant `LIMIT_ORDER_ACTION_ID`

D2: Fixed in commit `41470c6`

Cyfrin: Verified.

7.2.3 Parameter name mismatch between interface and implementation may be misleading

Description: In `IHype_Module`, the parameter names differ from the implementation:

- `hyper_sendSpot(uint64 token, uint64 _wei)` vs implementation uses `asset` for the first `uint64`
- `hyper_addApiWallet(address wallet, string calldata apiKey)` vs implementation uses `name` for the `string calldata`

The `apiKey` name is especially risky as it may lead someone to submit a secret API key on-chain, which would be permanently public.

Consider aligning interface and implementation parameter names (e.g., use `asset` and `name/label`).

D2: Fixed in commit `5c5cec4`

Cyfrin: Verified.

7.3 Gas Optimization

7.3.1 Remove loop when sending actions

Description: `Hype_Module::sendAction` manually allocates and copies a 4-byte header plus payload in a loop. Use packed encoding to avoid the loop and shrink bytecode, e.g.:

```
bytes memory data = abi.encodePacked(bytes4(uint32(0x01000000) | uint32(actionIndex)), action);
```

This builds the prefix + payload in one go with lower gas and less code.

D2: Fixed in commit c5d3193

Cyfrin: Verified.

7.3.2 `Hype_module::assetAddress` can be `pure`

Description: `Hype_module::assetAddress` neither reads nor writes state. Mark it `internal pure` to enable compiler optimizations, allow static analysis/staticcall-style usage, and slightly reduce gas/bytecode size while preventing accidental state access.

D2: Fixed in commit a217930

Cyfrin: Verified.