Obsidian
AUDITS

dHEDGE Security Review

Auditors

0xjuaan

0xSpearmint

March 18, 2026

Introduction

Obsidian Audits

Obsidian audits is a team of top-ranked security researchers, with a publicly proven track record, specialising in DeFi protocols across EVM chains and Solana.

The team has achieved 10+ top-2 placements in audit competitions, placing 1st in competitions for Wormhole, Pump.fun, Yearn Finance, and many more.

Find out more: obsidianaudits.com

Audit Overview

dHEDGE is a decentralized non-custodial tokenized vault protocol

dHEDGE engaged Obsidian Audits to perform a security review of the Hyperliquid integration contracts within the `dhedge-v2` repo. The review was conducted from March 16, 2026 to March 18, 2026. A subsequent review was conducted from March 26, 2026 to March 27, 2026, covering an update to the contracts to support HIP-3 trading.

Scope

Files in scope

Repo	Files in scope
`dhedge-v2` **Commit hash:** 92f4fedbbfc07b924ee4828b634f22acd5a9fe67	- guards/contractGuards/hyperliquid/HyperliquidCoreWriterContractGuard.sol - guards/contractGuards/hyperliquid/HyperliquidCoreDepositWalletContractGuard.sol - guards/assetGuards/hyperliquid/ HyperliquidPositionGuard.sol - guards/assetGuards/hyperliquid/ HyperliquidSpotGuard.sol - priceAggregators/ HyperliquidSpotPriceAggregator.sol - utils/hyperliquid/InFlightTracker.sol - utils/hyperliquid/PrecompileHelper.sol All paths relative to `contracts/`

The subsequent review (March 26–27, 2026) covered an update to the contracts to support HIP-3 trading: PR #1281 (commit `7d9a3a5`).

Summary of Findings

Severity Breakdown

A total of **8** issues were identified and categorized based on severity:

- **3 High severity**
- **1 Medium severity**
- **2 Low severity**
- **2 Informational**

Findings Overview

ID	Title	Severity	Status
H-01	Manager can steal depositor funds by disabling a supported asset after sending a market buy order	High	Fixed
H-02	Manager can steal funds by disabling a spot asset after bridging from a HIP-3 dex to the spot dex	High	Fixed
H-03	Manager can deflate totalFundValue by bridging tokens from HyperCore to HyperEVM and removing the asset in the same block	High	Fixed
M-01	Silent truncation in `_getPrice` causes slippage checks against wrong oracle price	Medium	Fixed
L-01	HYPE cannot be bridged between HyperCore and HyperEVM	Low	Acknowledged
L-02	HyperCore account activation of the pool is not enforced	Low	Fixed
I-01	If unified account mode is enabled, then bridging from Core to HyperEVM will not be possible	Informational	Fixed
I-02	Incorrect natspec comments	Informational	Fixed

Severity Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High -** leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium -** leads to a moderate material loss of assets in the protocol or moderately harms a group of users. Alternatively, breaking a core aspect of the protocol's intended functionality
- **Low -** leads to a minor material loss of assets in the protocol or harms a small group of users.

Likelihood

- **High -** attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- **Medium -** the attack/vulnerability requires minimal or no preconditions, but there is limited or no incentive to exploit it in practice
- **Low -** requires highly unlikely precondition states, or requires a significant attacker capital with little or no incentive.

Findings

[H-01] Manager can steal depositor funds by disabling a supported asset after sending a market buy order

Description

The `HyperliquidCoreWriterContractGuard` allows pool managers to place spot buy orders on HyperCore via the CoreWriter contract.

Orders sent from HyperEVM via CoreWriter are processed asynchronously by HyperCore, the order is submitted but not yet executed within the same EVM block. Precompile reads (such as `spotBalance`) reflect top-of-block state and will not include the results of any orders sent during the current block.

A manager can exploit this to manipulate the pool's `totalFundValue`:

1. The manager places an IOC spot buy order on HyperCore, spending USDC to buy a spot token.
2. In the same block, the manager calls `changeAssets` to remove the spot token from the pool's supported assets. The order has not yet executed on HyperCore, so `assetBalance` returns zero and the `_removeAsset` check passes.
3. Next block, the USDC spent becomes visible (reducing `totalFundValue`), but the acquired spot tokens are excluded from `totalFundValue` since the asset is no longer in `supportedAssets`.
4. The manager deposits into the pool at the deflated share price, receiving more shares than deserved.
5. The manager re-adds the spot token via `changeAssets`. The spot balance is now visible via the precompile, and `totalFundValue` restores to its true value.
6. After the withdrawal cooldown expires, the manager redeems their shares at the restored share price for profit.

The attack requires a supported spot token with a zero balance, for example a newly added token.

Recommendation

Consider recording when a spot buy order is placed and enforce a minimum delay (at least one block must pass) before the corresponding asset can be removed, ensuring the order has executed on HyperCore and the precompile state updates.

In the following block, the precompile state will have updated and `assetBalance` will return the non zero spot token balance, causing `_removeAsset` to correctly revert.

Remediation: Fixed in commit 326e5a6

[H-02] Manager can steal funds by disabling a spot asset after bridging from a HIP-3 dex to the spot dex

Description

In the `HyperliquidCoreWriterContractGuard`, in the guard for the `sendAsset` CoreWriter action, `_spotActionBlockNumber[poolLogic]` is only updated when `isBeingBridgedToEVM` is true.

```
if (isBeingBridgedToEVM) {
    // Record the block number when a spot asset bridging is placed for the
pool to disallow
    // removal of any spot asset from the supported assets list until the
bridging is completed.
    _spotActionBlockNumber[poolLogic] = block.number;
}
```

This means that if `sendAsset` is used to move assets from a HIP-3 dex to the spot dex, the `_spotActionBlockNumber[poolLogic]` mapping is not updated. This enables a manager to perform the following exploit:

1. Call `sendAsset` to queue an action to move a quote asset (like USDH) from a HIP-3 dex to the spot dex.
2. Call `changeAssets` to remove the spot USDH asset from the list of assets (this will succeed since the USDH balance is still 0)

As a result, in the next block, the spot USDH will be received but not accounted for in the NAV, allowing the manager to mint shares at a cheaper rate, which can be later redeemed for more assets once the manager calls `changeAssets` to support the spot asset again.

Recommendation

Consider ensuring that the `_spotActionBlockNumber[_poolLogic]` is updated to `block.number` whenever the `sendAsset` CoreWriter action occurs, not only when it's used for bridging to EVM.

Remediation: Fixed in commit 01038e4

[H-03] Manager can deflate totalFundValue by bridging tokens from HyperCore to HyperEVM and removing the asset in the same block

Description

The `HyperliquidCoreWriterContractGuard` allows pool managers to bridge tokens from HyperCore to HyperEVM. HyperCore actions occur asynchronously, therefore the EVM balance does not increase until the following block.

A manager can exploit this to deflate `totalFundValue` in two scenarios:

Scenario A: USDC

USDC on HyperEVM uses `ERC20Guard`, whose `getBalance` is simply `balanceOf(pool)`, while core USDC is tracked separately by `HyperliquidPositionGuard`.

1. The manager calls spot send to bridge USDC from HyperCore to HyperEVM.
2. In the same block, the manager removes USDC from supported assets. The EVM USDC has not arrived, so `ERC20Guard.getBalance` returns zero and removal passes.
3. Next block, core USDC decreases (reducing `HyperliquidPositionGuard.getBalance`) and the bridged USDC arrives on HyperEVM, but is excluded from `totalFundValue`.

Scenario B: Spot tokens with dual tracked system address and EVM contract

When both the system address and EVM contract of a spot token are supported assets, `HyperliquidSpotGuard.getBalance` excludes the core balance for the EVM contract entry to avoid double-counting:

```
uint256 coreBalance =
(IHasSupportedAsset(...).isSupportedAsset(getSystemAddress(tokenIndex)))
    ? 0
    : getCoreBalance(pool, tokenIndex);
```

1. The manager calls spot send to bridge the token from HyperCore to HyperEVM.
2. In the same block, the manager removes the EVM contract asset. Since getBalance excludes core balance and the EVM tokens have not arrived, it returns zero. Removal passes.
3. Next block, the system address balance drops (core balance decreased) and the arriving

EVM tokens are not counted.

In both scenarios, the manager deposits at the deflated share price, re-adds the asset, and withdraws at the restored price for profit.

Recommendation

This is a variant of [H-01] with a similar attack path.

The contracts should record when a bridging action occurs via spot send (action ID 6) or send asset (action ID 13), and check during `removeAssetCheck` that at least one block has passed since the last bridging action, reverting otherwise.

Remediation: Fixed in commit 57a4c6a

[M-01] Silent truncation in `_getPrice` causes slippage checks against wrong oracle price

Description

The `HyperliquidCoreWriterContractGuard` validates trade slippage by fetching the on-chain oracle price via precompiles and comparing it against the manager's submitted `limitPx`. For HIP-3 perps, the asset ID must be converted to a perp index (`assetId - 100_000`) before querying the oracle precompile.

In `_getPrice`, this derived perp index is cast to `uint16` before being passed to `normalizedOraclePx`. Solidity silently truncates the higher bits on downcasting. HIP-3 dexes with ID >= 7 produce perp indices above 65,535 (the `uint16` maximum). For example, dreamcash (dex ID 7) has perp indices starting at 70,000, which truncates to 4,464.

The slippage check then validates the manager's `limitPx` against the oracle price of a completely different asset. Depending on the price difference between the intended and truncated assets, this either blocks valid trades or more critically allows trades with arbitrary slippage to pass validation.

Recommendation

Change `perpIndex` in `_getPrice` and its associated casts from `uint16` to `uint32` to avoid silent truncation for HIP-3 perp indices that exceed 65,535.

```
- uint16 perpIndex = (assetType == AssetType.CORE_PERP) ? uint16(assetId) : uint16(assetId - 100_000);
+ uint32 perpIndex = (assetType == AssetType.CORE_PERP) ? uint32(assetId) : uint32(assetId - 100_000);
```

Remediation: Fixed in commit 7d9a3a5

[L-01] HYPE cannot be bridged between HyperCore and HyperEVM

Description

HYPE can be bought and sold on HyperCore via spot orders, but cannot be bridged in either direction between HyperCore and HyperEVM.

Core to EVM: The SpotSend action in `HyperliquidCoreWriterContractGuard` checks `isSupportedAsset(tokenInfo(params.token).evmContract)` to validate the asset. HYPE is the native gas token on HyperEVM, not an ERC20, so `tokenInfo` returns `address(0)` for its `evmContract`. `isSupportedAsset(address(0))` will return false, so the check reverts.

EVM to Core: Bridging HYPE back to HyperCore requires sending native value to the HYPE system address (`0x2222222222222222222222222222222222222222`). The `HyperliquidSpotGuard.txGuard` only handles ERC20 `transfer` and `approve` selectors. Additionally, `PoolLogic.execTransaction` uses `tryAssemblyCall` which hardcodes `value: 0` in the assembly call, making it impossible to send native value through any guard path.

Recommendation

If there is a need to bridge HYPE from Core to EVM, consider adding a system address fallback to the SpotSend asset check, consistent with the spot order check. For EVM to Core, add a guard path that supports sending native value to the HYPE system address.

Remediation: Issue acknowledged

[L-02] HyperCore account activation of the pool is not enforced

Description

When a pool bridges assets to HyperCore, the guards do not verify that the pool's HyperCore account is activated. If the account is not activated, bridged assets may not appear in `spotBalance`, causing them to become invisible to the guards' balance accounting. Note: this issue arises for depositable assets not including USDC/USDH, since bridging these to HyperCore will automatically activate the account on HyperCore (deducting the 1 USD fee).

The flow:

1. Pool bridges tokens to Core → `InFlightTracker` covers the gap temporarily.
2. Composite block advances → `getInFlightAmount()` returns 0.
3. `spotBalance` still returns 0 because the account is not activated.
4. Assets are invisible to both `HyperliquidPositionGuard.getBalance()` and `HyperliquidSpotGuard.getBalance()` → pool NAV is understated.

Recommendation

Implement `IAddAssetCheckGuard` in `HyperliquidSpotGuard`, using the `coreUserExists` precompile to verify the pool's HyperCore account is activated before allowing the asset to be added. This leverages the existing `addAssetCheck` hook called by `PoolManagerLogic._addAsset`, blocking misconfiguration before `getBalance` ever returns incorrect values.

Remediation: Fixed in commit f0960a1

[I-01] If unified account mode is enabled, then bridging from Core to HyperEVM will not be possible

Description

In unified account mode, the `spotSend` HyperCore action does not work (as seen here), and `sendAsset` (action ID 13) must be used to bridge assets from Core back to EVM. To bridge Core → EVM, the destination address must be the **system address** of the token (with `sourceDexId == spot` and `destinationDexId == spot`). However, the guard requires `destinationAddress == poolLogic`, which blocks this:

```
} else if (actionId == 13) {
    bytes memory actionParams = getParams(actionData);
    SendAssetParams memory params = abi.decode(actionParams,
(SendAssetParams));

    require(params.destinationAddress == poolLogic, "invalid destination
addr");
    require(params.subAccountAddress == address(0), "invalid sub-account
addr");
    require(
        params.destinationDexId == _DEX_ID_CORE_PERP ||
params.destinationDexId == _DEX_ID_CORE_SPOT,
        "invalid destination dex"
    );

    return
(uint16(ITransactionTypes.TransactionType.HyperliquidSendAssetAction),
false);
}
```

In the future, if unified account mode was enabled, there would be no guard-approved path to bridge assets back to the EVM side, so assets bridged to core would be stuck.

Recommendation

For future reference, if unified account mode is enabled for the HyperCore account of a vault, consider adjusting the `HyperliquidCoreWriterContractGuard` to allow `destinationAddress` to be the system address of the token being bridged (when both dex IDs are `_DEX_ID_CORE_SPOT`):

```
require(
    params.destinationAddress == poolLogic
+       || params.destinationAddress == getSystemAddress(params.token),
    "invalid destination addr"
);
```

Remediation: Fixed in commit de810db

[I-02] Incorrect natspec comments

Description

`PrecompileHelper.spotInfo` (line 191): natspec says "Fetches perp asset information" - should say "Fetches spot pair information".

`PrecompileHelper.spotPx` (line 318): natspec says "Fetches perp asset information" - should say "Fetches spot price".

`PrecompileHelper.oraclePx` (line 327): natspec says "Fetches perp asset information" - should say "Fetches oracle price".

`HyperliquidCoreWriterContractGuard._getPrice` (line 274): natspec says "the precompiles will revert for HIP-3 assets" - precompiles work for HIP-3 assets using the derived perp index. Remove the stale comment.

Remediation: Fixed in commit 7d9a3a5