Obsidian AUDITS

Bounce Tech Security Review

Auditors

0xjuaan

0xSpearmint

9th January, 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

Bounce Tech is a leveraged token protocol, built on HyperEVM.

Bounce Tech engaged Obsidian Audits to perform a security review of the smart contracts in the `bounce-tech/bounce-contracts` repo. The scope includes an update to support HyperEVM's native USDC as a base token. The review was conducted from the 3rd to the 4th of January.

Scope

Files in scope

Repo	Files in scope
`bounce-contracts` **Commit hash:** c05b9f40d98de2cc517badccca310344ac278ddd	src/Factory.sol src/HyperliquidHandler.sol src/LeveragedToken.sol src/LeveragedTokenProxy.sol

Summary of Findings

Severity Breakdown

A total of **2** issues were identified and categorized based on severity:

- **1 High severity**
- **1 Informational**

Findings Overview

ID	Title	Severity	Status
H-01	Incorrect usage of `_increaseBlockBridging()` in `bridgeFromSpot()` and `bridgeFromPerp()`	High	Fixed
I-01	Redundant logic in `_bridgeFromSpot`	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] Incorrect usage of _increaseBlockBridging() in \`bridgeFromSpot()\` and \`bridgeFromPerp()\`

Description

The `bridgeFromSpot` and `bridgeFromPerp` functions include the following line:

```
_increaseBlockBridging(block.number + 1, amount_);
```

This was introduced because offchain polling showed that the EVM balance of USDC bridged from HyperCore to HyperEVM would only be reflected in the 2nd block after the `spotSend` CoreWriter action.

However, after testing this onchain, it has been demonstrated that the bridged tokens will be present in the 1st block after the CoreWriter action, so there would be no need to apply any offset to the existing `totalAssets()` calculations.

Impact

Due to calling `_increaseBlockBridging`, the exchange rate will be inflated in the block immediately after `bridgeFromSpot` and `bridgeFromPerp` will be called, allowing people to redeem the base asset (USDC) at an unfairly high exchange rate.

Recommendation

Consider removing the `_increaseBlockBridging(block.number + 1, amount_)` call to ensure that the LeveragedToken's exchange rate is correct.

Remediation: Fixed in commit 7b3447d

[I-01] Redundant logic in `_bridgeFromSpot`

Description

The _bridgeFromSpot function manually retrieves the token index and converts the EVM amount to a core amount before calling `CoreWriterLib.bridgeToEvm`.

A convenience overload of `bridgeToEvm` exists that accepts a token address and EVM amount directly, handling both the token index lookup and amount conversion in the library function itself.

Recommendation

Consider replacing the existing logic with a single call to the alternate version of `bridgeToEvm()`, passing the base asset address and EVM amount directly.

```diff
function _bridgeFromSpot(uint256 amount_) internal {
-    uint64 tokenIndex_ = PrecompileLib.getTokenIndex(address(_baseAsset()));
-    uint64 coreAmount_ = HLConversions.evmToWei(tokenIndex_, amount_);
-    CoreWriterLib.bridgeToEvm(tokenIndex_, coreAmount_, false);
+    CoreWriterLib.bridgeToEvm(address(_baseAsset()), amount_);
}
```

Remediation: Fixed in commit 2199398