Obsidian
AUDITS

Bounce Tech Security Review

Auditors

0xjuaan

0xSpearmint

7th November, 2025

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, interacting with HyperCore for efficient rebalancing of positions.

Bounce Tech engaged Obsidian Audits to perform a security review on a portion of the smart contracts in the `bounce-contracts` repo. The review was conducted from the 18th to the 22nd of October.

Scope

Files in scope

Repo	Files in scope
`bounce-contracts` **Commit hash:** f803d7c	src/Factory.sol src/HyperliquidHandler.sol src/LeveragedToken.sol src/LeveragedTokenProxy.sol

Summary of Findings

Severity Breakdown

A total of **6** issues were identified and categorized based on severity:

- **2 Medium severity**
- **2 Low severity**
- **2 Informational**

Findings Overview

ID	Title	Severity	Status
M-01	Missing slippage protection in prepare and execute redemptions	Medium	Acknowledged
M-02	Changing streaming fee will apply the new fee retroactively	Medium	Acknowledged
L-01	Users can lose access to funds if agents do not call `executeRedemptions()` for them	Low	Fixed
L-02	The `executeRedemptionFee` can be increased to arbitrarily high values, causing loss to users	Low	Fixed
I-01	Adding an API wallet in `setAgent()` may silently fail	Informational	Fixed
I-02	Lack of validation of LeveragedToken account activation on HyperCore	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

[M-01] Missing slippage protection in prepare and execute redemptions

Description

Unlike `redeem()`, which enforces a `minBaseAmount_` parameter for slippage protection, the two-step redemption flow (`prepareRedeem()` + `executeRedemptions()`) does not include any minimum output check.

As a result, users who prepare redemptions have no guarantee on the final base amount they'll receive when agents later execute redemptions, which can be much lower than they expect due to price movements.

Recommendation

Consider adding a minimum acceptable base amount parameter or similar slippage safeguard to the two-step redemption process.

To fix this completely will also require applying the recommendation from L-01, so that users can can reclaim their LT tokens if the minimum amount out is not met for a long period of time.

Remediation: Issue acknowledged

[M-02] Changing streaming fee will apply the new fee retroactively

Description

The `streamingFee` is used to determine the fee paid over time by each `LeveragedToken`.

Changing the `streamingFee` in `GlobalStorage` will cause the new streaming fee to be used in the `LeveragedToken`, but this new fee will apply retroactively, for all fee accrued since the last accrual.

For example, assume `block.timestamp=100` and `lastCheckpoint=50`, and the streamingFee is now updated from 5% to 10%. Now, whenever `_checkpoint()` is called in the `LeverageToken`, the new `streamingFee` of 10% will apply starting from the timestamp of `50`, even though the fee from times `50` to `100` was 5%.

Recommendation

If this is unintended, consider adding a `lastStreamingFee` variable. And upon accrual, if `GlobalStorage.streamingFee()` is different to `lastStreamingFee`, then use `lastStreamingFee` for this accrual, and then update `lastStreamingFee` it to the new streamingFee.

This ensures that changes in the streaming fee are only applied proactively and not retroactively.

Remediation: Issue acknowledged

[L-01] Users can lose access to funds if agents do not call `executeRedemptions()` for them

Description

When a user calls the `prepareRedeem()` function, their LT tokens are transferred to the `LeveragedToken` contract and credited as a pending redemption. However, since only agents can finalize the process via `executeRedemptions()`.
If no agent ever executes the redemption (maliciously or otherwise), the user's funds remain locked, as there is no way for them to cancel their pending redemption to withdraw via `redeem()`.

Recommendation

Consider adding a `cancelRedeem()` function that allows users to reclaim their LT tokens after a defined timeout (e.g., 24 hours), if their redemption hasn't been executed.

Remediation: Fixed in PR #139

[L-02] The `executeRedemptionFee` can be increased to arbitrarily high values, causing loss to users

Description

The following constraint is used to enforce an upper bound on the `executeRedemptionFee`:

```
uint256 maxExecuteRedemptionFee_ =
minTransactionSize.mul(_MAX_EXECUTE_REDEMPTION_FEE_RATIO);
 if (executeRedemptionFee_ > maxExecuteRedemptionFee_) revert
InvalidAmount();
```

The issue is that the `minTransactionSize` can be arbitrarily increased by calling `setMinTransactionSize()`, meaning that the `executeRedemptionFee` can also be set to an arbitrarily high value like $500 USDT.

This means that in case the admin is compromised, the constraint on `executeRedemptionFee` can be bypassed, and an unfairly large fee can be set, causing loss to users upon withdrawals.

Recommendation

To mitigate this admin risk, consider setting an upper bound for `minTransactionSize` to ensure it cannot be set to arbitrarily high values. This will also ensure an upper bound for the `executeRedemptionFee`.

Remediation: Fixed in PR #138

[I-01] Adding an API wallet in `setAgent()` may silently fail

Description

The `setAgent()` function in the `LeveragedToken` contract calls the `addApiWallet` CoreWriter action to set an API wallet.

This action will fail to execute (without reverting in the EVM) if the API wallet is an activated account on HyperCore. As a result, the `setAgent()` call would succeed even though the `agent` was not actually added as an agent wallet.

Recommendation

Consider using the `coreUserExists` precompile to ensure that the agent's address is not activated on HyperCore, and to revert in case the account is activated.

Remediation: Fixed in PR #137

[I-02] Lack of validation of LeveragedToken account activation on HyperCore

Description

In order for CoreWriter actions to succesfully execute for the LeveragedToken, it's account must be activated on HyperCore (by transferring a small amount of any token to it, and paying a 1 USD fee). Without this, the `addApiWallet` CoreWriter action will silently fail.

Recommendation

In the `setAgent()` function, consider adding a check to ensure that `PrecompileLib.coreUserExists(address(this))` returns true. This would enforce the account is properly activated for all future CoreWriter actions.

Remediation: Fixed in PR #136