Security Assessment

ether.fi - ETHFI BeHYPE Audit Report

August 2025

Prepared for ether.fi

Table of contents

Project Summary

Project Scope

Project Name	Initial Commit Hash	Final Commit Hash	Platform	Start Date	End Date
beHYPE	Link	Link	EVM	26/08/2025	29/08/2025

Project Overview

This document describes the manual review of the live contract code for the **ETHFI beHYPE** project.

The work was a 4 day effort undertaken between **26/08/2025** and **29/08/2025**

The following contract list is included in our scope:

- `src/BeHYPE.sol`
- `src/BeHYPETimelock.sol`
- `src/RoleRegistry.sol`
- `src/StakingCore.sol`
- `src/WithdrawManager.sol`
- `src/lib/BucketLimiter.sol`
- `src/lib/CoreWriter.sol`
- `src/lib/L1Read.sol`
- `src/lib/UUPSProxy.sol`

The team performed a manual audit of the deployed Solidity smart contracts. During the manual audit, the Certora team discovered bugs in the Solidity smart contracts code, as listed on the following pages.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	1	1	1
High	1	1	–
Medium	1	1	–
Low	5	5	4
Informational	10	10	5
Total	18	18	9

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Detailed Findings

ID	Title	Severity	Status
C-01	Any staker can permanently block withdrawals	Critical	Fixed
H-01	Insolvency due to slashing risks or commission charging	High	Acknowledged
M-01	Frontrunning ratio changes from updateExchangeRatio()	Medium	Acknowledged
L-01	depositToHyperCore() may cause loss of funds due to truncation	Low	Fixed
L-02	Deadlock due to minWithdrawalAmount	Low	Acknowledged
L-03	Missing slippage protection on withdraw	Low	Fixed
L-04	CEI pattern not followed upon withdrawal finalization	Low	Fixed
L-05	L1 latency may cause incorrect share ratio	Low	Fixed

Critical Severity Issues

C-01 Any staker can permanently block withdrawals

Severity: **Critical**	Impact: **High**	Likelihood: **High**
Files: WithdrawManager.sol	Status: Fixed	

Description: finalizeWithdrawals() takes the index of the latest withdrawal to be finalized and iterates through all the pending withdrawals up to that index:

```
JavaScript
function finalizeWithdrawals(uint256 index) external {
        ....
        if (index >= withdrawalQueue.length) revert IndexOutOfBounds();
        if (index <= lastFinalizedIndex) revert CanOnlyFinalizeForward();
        ....
        for (uint256 i = lastFinalizedIndex + 1; i <= index;) {
             //@audit - can DOS withdraws
            stakingCore.sendFromWithdrawManager(withdrawalQueue[i].hypeAmount,
withdrawalQueue[i].user);
            ...
```

However, on each iteration it will call sendFromWithdrawManager , which invokes the following:

```
JavaScript
function sendFromWithdrawManager(uint256 amount, address to) external {
        if (msg.sender != withdrawManager) revert NotAuthorized();

        (bool success,) = payable(to).call{value: amount}("");
        if (!success) revert FailedToSendFromWithdrawManager();
    }
```

As a result, an attacker can revert the Hype transfer in their receive() function and block all other users from withdrawing.

Recommendations:

Consider introducing the following changes in finalizeWithdraw():

- Transfer the hypeAmount to the withdrawManager contract, instead of each user.

- Allocate the claimable user amounts in a new mapping - claimableBalance.

- Create a claim() function that users will use to claim their hype tokens and update the claimableBalance mapping.

Customer's response: Fixed in commit [04fd66ceca]

Fix Review: Fixed

High Severity Issues

H-01 Insolvency due to slashing risks or commission charging

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: WithdrawManager.sol	Status: Acknowledged	

Description: In the WithdrawManager there is a two-step withdrawal flow. Initially, users will trigger the withdraw() function, which will **lock in** the Hype amount based on the current ratio :

```
JavaScript
  uint256 hypeAmount = stakingCore.BeHYPEToHYPE(beHypeAmount);
```

This is done so that withdrawing shares can stop receiving rewards, when a withdrawal is initiated. However, this will also allow users to completely reduce their slashing risk and commission charges from the validator, as their Hype amount will already be fixed. Furthermore, in some edge cases it can also result in insolvency as the StakingCore may not have enough assets due to any of the charges on the Core.

Rewards will also be subject to dilution as the total supply will remain unchanged for the period between the initial withdraw() invocation and the finalizeWithdrawals() call. As a result, anytime there are pending withdrawals, the following ratio calculation will incorrectly allocate rewards equally to all shares, temporarily decreasing the rewards for active users.

```
JavaScript
 uint256 newRatio = Math.mulDiv(totalProtocolHype, 1e18, beHypeToken.totalSupply());
```

Recommendations: Consider refactoring the code in a way to properly isolate withdrawals from the active users' supply and tracking the withdrawals independently. For example, decrease the total protocol Hype by the pending withdrawal Hype and burn the shares during the withdraw()

call. In case of a detected slashing, the pendingWithdrawalAmount can be decreased, based on the ratio change, so that users in the queue are also charged.

Customer's response: Acknowledged - *"Hyperliquid has no automated slashing implemented making this issue highly unlikely*

If a large-scale consensus attack occurs, Hyperliquid could use social layer mechanisms to penalize malicious staking. In this event we will pause withdrawals and withdrawal finalization to allow this process to take place."

Fix Review: Acknowledged

Medium Severity Issues

M-01 Frontrunning ratio changes from updateExchangeRatio()		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: StakingCore.sol	Status: Acknowledged	

Description: The ratio is updated only using the updateExchangeRatio(), which will be called by the admin. However, this introduces front-running risks:

- Users can front-run updateExchangeRatio() and instantly withdraw before a slashing or commission is applied.
- Users can front-run updateExchangeRatio() and stake before a big amount of rewards is allocated.

Recommendations: One way to reduce the incentive for frontrunning attacks is to introduce a staking fee.

Customer's response: Acknowledged- *"We realize the risk here, but see the instant withdrawal fee and forgoing of staking required for standard withdrawals as a sufficient deterrent"*

Fix Review: Acknowledged

Low Severity Issues

L-01 depositToHyperCore() may cause loss of funds due to truncation

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: StakingCore.sol	Status: Fixed	

Description: In HyperEVM, Hype uses 18 decimals while in HyperCore only 8 decimals are used. When transferring across the bridge, any precision beyond 8 decimals is truncated. As a result the following may cause loss of funds due to this truncation:

JavaScript

```
function depositToHyperCore(uint256 amount) external {
      ....
       //@audit truncation
       (bool success,) = payable(L1_HYPE_CONTRACT).call{value: amount}("");
       ....
    }
```

If amount has value beyond the 8 decimals it would be lost

Recommendations: Validate that amount does not hold any value after the 8 decimals before sending it to L1_HYPE_CONTRACT

Customer's response: Fixed in commit f312086cf

Fix Review: Fixed

L-02 Deadlock due to minWithdrawalAmount

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: WithdrawManager.sol	Status: Acknowledged	

Description: The minWithdrawalAmount variable prevents withdrawing small amounts of beHype. However it does not enforce deadLock protection, which means that users whose balance falls below this minWithdrawalAmount will be unable to withdraw and need to further deposit and expose themselves to unwanted risk, in order to reach the threshold to be able to withdraw.

Recommendations: In case the full beHype balances of an account are below minWithdrawalAmount, withdrawal should be allowed if they are requested in full - e.g. the account beHype balances would be reduced to 0

Customer's response: Acknowledged - *"We consider this an acceptable trade-off to prevent spam on our withdrawal queue as deadlocked users can utilize DEX liquidity to swap their BeHYPE tokens if needed, providing an alternative exit mechanism without requiring protocol modification"*

Fix Review: Acknowledged

L-03 Missing slippage protection on withdraw

Severity: **Low**	Impact: **Medium**	Likelihood: **Medium**
Files: WithdrawManager.sol	Status: Fixed	

Description: The withdraw function will redeem shares based on the beHypeAmount provided and the current BeHYPEToHYPE ratio. However this ratio can be updated anytime by the owner, by calling updateExchangeRatio() . Due to the nature of the blockchain, potential transaction latency may result in an unexpected hypeAmount received, especially for instant withdrawals.

Recommendations: Consider adding a minAmountOut parameter for the withdraw function.

Customer's response: Fixed in commit 263872e0b

Fix Review: Fixed

L-04 CEI pattern not followed upon withdrawal finalization

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: WithdrawManager.sol	Status: Fixed	

Description: The finalizeWithdrawals() function executes a loop, where each iteration contains a re-entrant call through native token transfer:

```
JavaScript
for (uint256 i = lastFinalizedIndex + 1; i <= index;) {
             //@audit- re-entrant
            stakingCore.sendFromWithdrawManager(withdrawalQueue[i].hypeAmount,
withdrawalQueue[i].user);
            beHypeAmountToFinalize += withdrawalQueue[i].beHypeAmount;
            hypeAmountToFinalize += withdrawalQueue[i].hypeAmount;
            withdrawalQueue[i].finalized = true;

            unchecked { ++i; }
        }
```

The issue is that all the beHypeToken are burned only at the end of the function (after the HYPE transfers). This creates an intermediate state where HYPE tokens are sent to their account owners, while all their beHYPE shares are not burned. Currently the exchange ratio is not dynamic (updated by the protocol admin) and this is not exploitable, but if it is changes to a dynamic an auto-updating ratio based on staked balances, it could allow calling stake (non-protected) in that intermediate state leading to excessive shares being minted to the detriment of other stakers

Recommendations: It is recommended to always execute the re-entrant call after the state update, to guarantee the logic would be future proof and not allow any unexpected loop-holes.

Burn the respective beHypeAmount on each iteration of the loop **before** the HYPE transfer. Although more gas intensive this prevents the issues explained above

Customer's response: Fixed in commit 04fd66ceca

Fix Review: Fixed

L-05 L1 latency may cause incorrect share ratio

Severity: **Low**	Impact: **High**	Likelihood: **High**
Files: StakingCore.sol	Status: Fixed	

Description: On the L1, each block reads the Core state for the end of the previous block. This creates a potential issue in which updateExchangeRatio returns an incorrect ratio due to unprocessed deposits to the spot balance.

For instance, there could be 50e18 Hype currently in the StakingCore. If the admin calls depositToHyperCore() in block 7, passing 20 tokens, and also executes updateExchangeRatio() in that block, these 20 tokens will not be read because they are locked on the system contract on the EVM and they are also not yet assigned on the core spot balances. An analogical issue also exists for withdrawFromHyperCore().

Recommendations: Consider saving the block.number when depositToHyperCore() and withdrawFromHyperCore() have been invoked and ensuring a minBlocksPassed duration, in which the updateExchangeRatio() will not be callable, so that a proper state is fetched.

Customer's response: Fixed in commit 096931f6

Fix Review: Fixed

Informational Issues

I-01. Emit events on state changing functions

Description: No events are emitted for the following state changing functions:

- StakingCore.updateAcceptableApr()
- StakingCore.updateExchangeRateGuard()
- WithdrawManager.setInstantWithdrawalCapacity()
- WithdrawManager.setInstantWithdrawalRefillRatePerSecond()

Recommendation: Emit events in the above functions

Customer's response: Fixed in commit 525a3bf3

Fix Review: Fixed

I-02. Wrong revert message in withdrawFromHyperCore()

Description: NotAuthorized shall be replaced by ExceedsLimit or a similar error message for the following validation check in withdrawFromHyperCore :

JavaScript

```
function withdrawFromHyperCore(uint amount) external {
    ....
    if (amount > IWithdrawManager(withdrawManager).hypeRequestedForWithdraw()) revert
NotAuthorized();
}
```

Recommendation: Add proper error

Customer's response: Fixed in commit fb1c2d8cb3

Fix Review: Fixed

I-03. Rename parameter names

Description: The following functions use wrong parameter names:

- StakingCore.BeHYPEToHYPE() - it uses kHYPEAmount, while it should be beHYPEAmount
- WithdrawManager.initialize() - _min/maxStakeAmount should be renamed to _min/maxWithdrawAmount

Recommendation: Implement the above recommendations

Customer's response: Fixed in commit 6a5588f3da

Fix Review: Fixed

I-04. Redundant check in withdrawFromHyperCore()

Description:
The following amount check can be easily bypassed by calling the function multiple times with smaller amounts. Furthermore, a portion of the requested Hype may already be in the contract.

JavaScript

```
function withdrawFromHyperCore(uint amount) external {
    ....
    if (amount > IWithdrawManager(withdrawManager).hypeRequestedForWithdraw()) revert
NotAuthorized();
}
```

Recommendation: Consider removing the check

Customer's response: Fixed in commit fb1c2d8cb3a

Fix Review: Fixed

I-05. withdraw() will round down the withdrawal fee

Description:
The withdraw function uses the following to compute the instantWithdrawalFee:

JavaScript

```javascript
 uint256 instantWithdrawalFee = beHypeAmount.mulDiv(instantWithdrawalFeeInBps,
BASIS_POINT_SCALE);
```

Recommendation: Consider rounding the fee up.

Customer's response: Acknowledged – *"This behavior aligns with our business needs"*

Fix Review: Acknowledged

I-06. Vault inflation attack

Description:
An attacker could artificially inflate the vault's share price by first depositing a minimal amount (e.g., **1 wei**) and then donating a disproportionately large amount of assets (e.g., **1e18 units**) directly to the vault. When the privileged `updateExchangeRatio()` function is subsequently invoked by the administrator, the vault's share price would be recalculated based on the inflated asset-to-share ratio. This could cause rounding or truncation issues for subsequent deposits, particularly for users depositing amounts with fewer than **18 decimal places** of precision. This attack is extremely limited due to the updateExchangeRatio() access control and the exchange rate guard.

Recommendation: Consider introducing virtual shares or an initial stake so that the ratio cannot be easily inflated.

Customer's response: Acknowledged - *"We are aware of this risk but consider the risk acceptable given attack is extremely limited due to the updateExchangeRatio() access control and the exchange rate guard"*.

Fix Review: Acknowledged

I-07. Sanity checks

Description: Consider adding the following checks to make validation more robust and ensure protocol variables cannot be configured with unexpected values:

- Prevent address(0) being assigned in the following scenarios:
  - Inside BeHype.initialize() for roleRegistry, stakingCore & withdrawManager
  - Inside RoleRegistry.initialize() for _owner, withdrawManager, stakingCore & protocolTreasury , also in setProtocolTreasury(), setWithdrawManager() & setStakingCore() as well
  - Inside StakingCore.initialize() for roleRegistry, beHypeToken & withdrawManager and also in setWithdrawManager()
  - Inside StakingCore.delegateTokens() make sure validator is not address(0)
  - Inside WithdrawManager.initialize() for roleRegistry, beHypeToken, stakingCore
- Inside WithdrawManager.initialize() check that _minStakeAmount/_maxStakeAmount are < BASIS_POINT_SCALE
- Put a hardcoded max limit on the value passed inside WithdrawManager.setInstantWithdrawalFeeInBps(), to ensure the protocol fee cannot be set to unreasonably high value

Recommendation: Implement the above validation checks

Customer's response: Acknowledged - *"We consider the risk acceptable given deploy scripts and any updates made by PROTOCOL_GUARDIAN undergo a strict review process"*

Fix Review: Acknowledged

I-08. The annualized extrapolation of ratioChange can be sensitive to short timeframe changes

Description: The ratioChange check in updateExchangeRatio computes the absolute difference between ratios based on elapsedTime and then annualizes it by scaling with (365 days / elapsedTime). This goal of the check is to protect from rapid and unexpected swings in the exchange rate.

One specific detail in that approach is that the calculation can be very sensitive if the function is called for very short time frames, since the result would be “exaggerated” multiple times (up to 365 days).

Recommendation: The team should be aware of that and make sure to test and schedule the updateExchangeRatio calls to be called in proper intervals, so that that the ratio check would behave as expected.

Customer’s response: Acknowledged

Fix Review: Acknowledged

I-09. Anyone can claim on behalf of other users

Description: Currently the claimWithdrawal() function allows any caller to invoke the claim transfer from the WithdrawManager to the user who initiated the original request. This may be problematic for contracts which integrate with the BeHype, due to potential unexpected claims.

Recommendation: Consider adding access control, so that only the user who initiated the withdrawal request can claim it.

Customer's response: Acknowledged- *"We believe the risk here is worth the ability to add a feature to claim on behalf of users for a more automated withdrawal process"*

Fix Review: Acknowledged

I-10. Emergency withdrawing from core staking balances is restricted by pending withdraws

Description: The newly implemented check in withdrawFromStaking() is meant to guard in case the PROTOCOL_ADMIN() role is compromised. It restricts the withdrawal amounts up to the hypeRequestedForWithdraw amounts. This creates another restriction where even the protocol multisig will not be able to emergency withdraw if there are no pending withdrawal requests

Recommendation: Consider if the above scenario is acceptable and if not add an additional multisig controlled function that allows emergency withdraws

Customer's response: Fixed in commit e80a2ca30

Fix Review: Fixed

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.