Security Assessment

ether.fi – beHYPE Bridging Contracts Combined Audit Report

November 2025

Table of contents

Project Summary

Project Scope

Project Name	Initial Commit Hash	Latest Commit Hash	Platform	Start Date	End Date
BeHYPE Scroll Staker	95904ab	1b05b52e	EVM	31/10/2025	04/11/2025
BeHYPE Staking Module	dbefde0	bdfb44f	EVM	06/11/2025	07/11/2025

Project Overview

This document describes the manual code review of several changes to the bridging contracts throughout the **cash-v3** & the **beHYPE** repositories.

The work was a 3 day effort undertaken between **31/10/2025** and **07/11/2025**

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

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	3	3	3
Medium	1	1	1
Low	3	3	3
Informational	2	2	1
Total	9	9	8

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
BeHYPE Scroll Staker			
H-01	TransferFrom can be front-runned to steal approved amounts	High	Fixed
H-02	Token amounts past the precision point would be lost	High	Fixed
H-03	Cross-chain staking leads to less BEHYPE being received on the destination chain	High	Partially Fixed
M-01	Cross-chain communication pattern is suboptimal and can often lead to inconsistencies and DOS	Medium	Fixed
BeHYPE Staking Module			
L-01	Incorrect handling of cross-chain fee funding (native token)	Low	Fixed
L-02	Lack of refund mechanism for excess cross-chain fees	Low	Fixed
L-03	Unused approval is not cleared	Low	Fixed

Project Overview

This report presents the findings of a manual code review for the **BeHYPE Scroll Staker** audit within the **EtherFi beHYPE** repository. The work was undertaken between **October 31st 2025** and **November 4th 2025**

The following contract list is included in the scope of this audit:

- `src/L1BeHYPEOAppStaker.sol`
- `src/L2BeHYPEOAppStaker.sol`

The code modifications examined during this review were implemented in the following pull request - PR#23

High Severity Issues

H-01 TransferFrom can be front-runned to steal approved amounts

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: L2BeHYPEOAppStaker.sol	Status: Fixed	

Description: The L2BeHYPEOAppStaker contract uses transferFrom() to transfer WHYPE inside the stake() function before sending a cross-chain message:

```
JavaScript
function stake(uint256 hypeAmountIn, address staker, address receiver) external payable {
        ....
         // Send the users WHYPE to the peer contract on hyperEVM
         IERC20(address(WHYPE)).transferFrom(staker, address(this), hypeAmountIn);
         WHYPE.send{value: params.oftFee.nativeFee}(params.oftParam, params.oftFee,
msg.sender);
        ....
    }
```

Problem here is that transferFrom() takes an arbitrary staker address, which allows a malicious user to front-run the approvals given to OAppStaker and send the amounts to another address controlled by the attacker, instead of the intended recipient.

Recommendation: Instead of using a caller provided address for transferFrom(), use msg.sender to ensure assets are always transferred from the sender and other approvals cannot be exploited.

Customer's response: Fixed in commit [6ddcbacb]

Fix Review: Fixed

H-02 Token amounts past the precision point would be lost

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: L2BeHYPEOAppStaker.sol	Status: Fixed	

Description: OFT tokens between two chains have shared decimals precision of 6 (in the default OZ implementation), which means that for tokens with 18 decimals everything past the 6 decimal point is considered dust amount and will not be sent by LZ. The following function ensures this behaviour in the OFT contract before calling _lzSend().

The L2BeHYPEOAppStaker does not consider this and has the following logic:

```
JavaScript
function stake(uint256 hypeAmountIn, address staker, address receiver) external payable {
....
// Send the users WHYPE to the peer contract on hyperEVM
        IERC20(address(WHYPE)).transferFrom(staker, address(this), hypeAmountIn);
        WHYPE.send{value: params.oftFee.nativeFee}(params.oftParam, params.oftFee,
msg.sender);
...
```

Currently this means that if hypeAmountIn is for example 1....99999999999(1e18) the WHYPE.send() call will zero out everything after the 6 decimal and only transfer that value.

There are a couple of serious effects from this:

- Users would constantly lose the token value past the precision point, because it will remain locked in the contract

- The bridged message on the receiving side will either be DOSed or the sender would stake to the detriment of other stakers - this is because the encoded message sent to L1BeHYPEOAppStaker uses hypeAmountIn which is bigger than the actual tokens being sent, because it is the initial amount before being cleaned from the excess value

Recommendation: Consider validating no truncation beyond the 6th decimal will occur.

Customer's response: Fixed in commit 6ddcbacb

Fix Review: Fixed

H-03 Cross-chain staking leads to less BEHYPE being received on the destination chain

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: L1BeHYPEOAppStaker.sol	Status: Partially fixed	

Description: Currently WHYPE is bridged from Scroll to HypeEVM into the L1BeHYPEOAppStaker contract where it is unwrapped into native HYPE, then staked for beHYPE, where the beHYPE is finally bridged back to Scroll.

The STAKING_CORE contract has a constantly changing exchange rate that determines the amount of beHYPE to receive based on the provided HYPE (WHYPE).

Issues here is that there is 6 decimal shared precision between beHYPE on HypeEVM & Scroll, which means that value of byHYPE past the 6 decimals will not be bridged and will remain locked in the contract, leading to the scroll recipient to receive less beHYPE.

This issue differs from the above one in that the beHYPE amounts are computed during execution (not user controlled) and they will be most of the time not precise, meaning that recipients would almost always lose on beHYPE tokens

Recommendation: For simplicity, consider adding a sweep function to recover the dust funds as fee. Additionally, consider documenting this behaviour as a design limitation.

Customer's response: Partially fixed in commit 6ddcbacb

Fix Review: Partially Fixed - *"The team has implemented a sweeping function to collect the accumulated leftover value from the last 12 decimals. Users bridging WHYPE would still partially*

lose value on BYHYPE past the precision point, so make sure to also state it in the official docs for the users."

Medium Severity Issues

M-01 Cross-chain communication pattern is suboptimal and can often lead to inconsistencies and DOS

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: L2BeHYPEOAppStaker.sol	Status: Fixed	

Description: The pattern currently used by the team to bridge tokens between Scroll and HypeEVM is the following:

1. L2BeHYPEOAppStaker is deployed on scroll where:
    a. One cross-chain call is sent to the WHYPE OFT contract to bridge to L1BeHYPEOAppStaker on HypeEVM
    b. A second independent cross-chain call is sent through _lzSend() to invoke lzReceive() on L1BeHYPEOAppStaker on HypeEVM, which stakes the WHYPE transferred in the above call

This approach deviates from the mechanism developed by LZ to handle such flows and is prone to race conditions which can lead to DOS.

The main issue here is that LZ bridging calls have no guarantee over the order they will get delivered by the executors to the destination chain.

For example if the second call that does the staking is delivered before the token is transferred it would fail. This would require the team to additionally pay gas and manually retry such failed messages which can be frequent

Recommendations: The recommended way to handle token bridging + subsequent actions on the destination chain is through the so called Composed OFT messages - the pattern is explained in depth here (transfer + staking is among the exact use cases).

Using composable messages also guarantees that token transfer will always happen first before the composable (staking) message gets executed in a separate transaction

This will also reduce complexity because it will use the already available functionality part of the default OFT implementation (WHYPE in this case)

Customer's response: Fixed in commit 6ddcbacb

Fix Review: Fixed

Informational Issues

I-01. No way to update enforceOptions

Description: The enforceOptions parameter is a state variable set upon initialization that is used to define the Executor options for the cross-chain message.

This is a dynamic value that might change with time, but currently there is no function for updating it

Recommendations: Introduce an additional function for updating enforceOptions

Customer's response: Fixed in commit 6ddcbacb

Fix Review: Fixed

I-02. Potential fee griefing in _lzReceive

Description: In _lzReceive() the fee is assumed to be pre-funded into the contract. However, since this is a singleton contract, all users will be able to use these funds. An attacker could spend an increased amount of that fee, by splitting their stakes. This attack is however disincentivized as in stake(), the attacker must also pay the oft.nativeFee as well as the lzFee.nativeFee.

Recommendations: This is an unprofitable and low likelihood vector, but it is still useful to consider it for future code changes.

Customer's response: Acknowledged

Fix Review: Acknowledged

BeHYPE Staking Module

Project Overview

This report presents the findings of a manual code review for the **BeHYPE Staking Module** audit within the **cash-v3** repository. The work was undertaken between **November 6th 2025** and **November 7th 2025**

The following contract list is included in the scope of this audit:

- `src/modules/hype/BeHYPEStakeModule.sol`

The code modifications examined during this review were implemented in the following pull request - PR#68

Low Severity Issues

L-01 Incorrect handling of cross-chain fee funding (native token)

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

Description:
The stake() function accepts msg.value as the cross-chain fee, but this native token is retained in the BeHYPEStakeModule contract. When execTransactionFromModule is executed, the Safe performs the call to staker.stake() using its own native token balance instead of the amount provided by the caller. If the Safe lacks sufficient native balance, the staker.stake() call will revert.

Recommendation:
Transfer the provided msg.value (equal to the quoted fee) from the module to the Safe before calling execTransactionFromModule, ensuring the Safe can fund the LayerZero fee.

Customer's response: Fixed in commit 0be1990

Fix Review: Fixed

L-02 Lack of refund mechanism for excess cross-chain fees

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: BeHYPEStakeModule.sol	Status: Fixed	

Description:
The stake() function requires the caller to send msg.value to cover the cross-chain LayerZero fee. However, if the caller provides more than the quoted fee, the excess amount is not refunded. This can lead to users unintentionally overpaying for transactions.

Recommendation:
Implement logic to refund any excess msg.value beyond the quoted cross-chain fee back to the caller after transferring the required fee to the Safe.

Customer's response: Fixed in commit 0be1990

Fix Review: Fixed

L-03 Unused approval is not cleared

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: BeHYPEStakeModule.sol	Status: Fixed	

Description: The BeHYPEStakeModule sends an approval call for the provided amountToStake, however the L2BeHYPEOAppStaker contract will clear the excess amount and might consume less - leaving unused approval from the Safe to L2BeHYPEOAppStaker

Recommendation: Consider explicitly resetting the approval to 0 after the bridging call, to make sure there are no unused approvals

Customer's response: Fixed in commit 0be1990

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.