MixBytes()

Euler StakeDelegator Security Audit Report

APRIL 22, 2025

Table of Contents

1. Introduction

1.1 Disclamer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

The HookTargetStakeDelegator contract is a hook implementation for EVault that enables delegated staking of vault shares. It tracks user balances change through the checkVaultStatus function and automatically mints ERC20 tokens representing the shares and stakes them in the Berachain RewardVault on behalf of the user.

This security audit was conducted over a 2-day period.

During the audit, in addition to checking well-known attack vectors and those listed in our internal checklist, we carefully investigated the following:

User balance tracking via hook. The HookTargetStakeDelegator hook implements comprehensive user balance tracking through its checkVaultStatus function. It maintains a touchedAccounts set to track all affected accounts in a transaction and uses an initialBalances mapping to record starting balances when accounts are first touched. The hook calculates net changes by comparing current balances with initial balances and properly handles both direct account ownership and EVC-controlled accounts through evc.getAccountOwner(account). The balance tracking is atomic within a transaction, ensuring no updates are missed.

Resilience against malicious blocking. We confirmed that the hook logic **cannot be blocked by a malicious user** to prevent the protocol from processing updates. The staking operations in _delegateStake and _delegateWithdraw are non-reentrant and protected by the RewardVault's security measures. The hook makes no external calls before state updates, preventing reentrancy attacks, and its operations are independent of user input, making it resistant to malicious user behavior.

Accurate and flexible hook flow. The hook's flow correctly accounts for **all possible balance update scenarios**, and it can be **opted into after Vault creation** without negatively impacting accounting. It correctly processes new deposits (positive net change), withdrawals (negative net change), multiple operations in the same transaction, and account ownership changes through EVC.

Berachain integration. We thoroughly reviewed the **integration with Berachain contracts** to ensure there is **no way to block protocol execution** through malicious behavior on the Berachain side. The integration is secure because token transfers are atomic with staking operations and account ownership is verified through EVC. The hook cannot be used to manipulate the RewardVault's state and is resilient to malicious token transfers and invalid stake operations.

The RewardVault upgradeability. The RewardVault contract inherits from upgradeable OpenZeppelin contracts. This upgradeability introduces potential security considerations that must be carefully managed. The upgrade process could potentially change the behavior of critical functions like delegateStake and delegateWithdraw. The hook's interaction with the RewardVault must be resilient to such upgrades, ensuring that the staking and withdrawal operations continue to function correctly even after contract upgrades.

Operation flag handling. The audit verified that the hook properly handles all implemented operation flags. We specifically examined the critical importance of ensuring that unsupported operation flags (OP_BORROW, OP_PULL_DEBT, OP_CONVERT_FEES, OP_LIQUIDATE, OP_FLASHLOAN, OP_TOUCH) are not enabled in the Hook's configuration. The audit confirmed that enabling these flags in the Hook's configuration would lead to unexpected reverts.

The audit was focused specifically on:
- Examining the token minting and staking flow security.
- Verifying the proper handling of account ownership changes through EVC.
- Assessing the protection against reentrancy attacks.
- Reviewing the security of the withdrawal process.
- Evaluating the handling of multiple operations within single transactions.
- Verifying the proper tracking of initial balances and net changes.
- Assessing the security of the touchedAccounts set management.
- Analyzing the security implications of the RewardVault's integration.
- Analyzing the impact of hook installation timing on share accounting, particularly how the initialBalances tracking affects existing shares when the hook is added to an already active Vault.

The HookTargetStakeDelegator contract demonstrates high code quality and robust security design. With only three medium-severity and one low-severity issue identified during the audit, the contract shows careful consideration of security best practices and proper implementation of critical functionality.

1.3 Project Overview

Summary

Title	Description
Client Name	Euler
Project Name	HookTargetStakeDelegator
Type	Solidity
Platform	EVM
Timeline	07.04.2025 - 16.04.2025

Scope of Audit

File	Link
src/HookTarget/HookTargetStakeDelegator.sol	HookTargetStakeDelegator.sol

Versions Log

Date	Commit Hash	Note
07.04.2025	995449e6e960b515869f00235fdcd417ad8e08fc	Initial Commit
14.04.2025	647866626fbceec678e3e11cdd9d7e5be9e5f6e5	Commit for the Re-audit
16.04.2025	491a657e865c9ad2e6ecd1c3c20f0443e0d68422	Commit with updates

Mainnet Deployments

File	Address	Blockchain
HookTargetStakeDelegator.sol	0x69CdFe...592417d7	Berachain
HookTargetStakeDelegator.sol	0x0Cb476...77D44874	Berachain
HookTargetStakeDelegator.sol	0xef452d...26F7B618	Berachain

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	1
Medium	3
Low	1

Findings Statuses

ID	Finding	Severity	Status
H-1	Double-counting of the Migrated Stake	High	Fixed
M-1	Incomplete Stake Migration Logic if Balance Only Increases	Medium	Fixed
M-2	Deposits Made Before Hook Connection are not Accounted for Rewards	Medium	Acknowledged
M-3	Stuck Rewards Due to Incorrect Stake Delegation	Medium	Acknowledged
L-1	Missing RewardVault Deployment Check During Stake Delegation	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

H-1	Double-counting of the Migrated Stake		
Severity	High	**Status**	Fixed in 491a657e

Description

There is an issue with HookTargetStakeDelegator.sol#L283 inside the _delegateStake function. _migrateStake internally delegates the stake from the account to the owner and returns the delegated amount. This amount is then added to the amount value originally passed to the _delegateStake function.

The issue is classified as **High** severity because it leads to double-accounting for the delegated value from the account to the owner address, which is incorrectly added to the original amount, which was subject to delegation.

Recommendation

We recommend changing the logic inside the internal _delegateStake function to the following lines:

```
function _delegateStake(
    address account,
    uint256 amount) internal {
    address owner = evc.getAccountOwner(account);

    _migrateStake(owner, account);
    rewardVault.delegateStake(
        owner == address(0) ? account : owner,
        amount);
}
```

This ensures that the migrated stake is not double-counted, and there is a maintained clear distinction between migration and new delegation.

Client's Commentary:

Fixed: PR-270

2.3 Medium

M-1	Incomplete Stake Migration Logic if Balance Only Increases		
Severity	Medium	**Status**	Fixed in 491a657e

Description

This issue has been identified within the checkVaultStatus() function of the HookTargetStakeDelegator contract.

Stake migration HookTargetStakeDelegator.sol#L296-L303 during the withdrawal process. As a result, if an account's balance consistently increases and no withdrawals occur, the delegated stake will remain split between the accounts and its registered EVC owner, rather than being fully migrated to the owner.

The issue is classified as **Medium** severity because it reduced effectiveness of the staking system.

Recommendation

We recommend modifying the checkVaultStatus() logic to ensure the stake migration also occurs during balance increases, not only during withdrawals.

Client's Commentary:

Fixed: PR-274

MixBytes: There is an incorrect accounting for the delegated stake. The issue is described in the H-1 above.

M-2	Deposits Made Before Hook Connection are not Accounted for Rewards		
Severity	Medium	**Status**	Acknowledged

Description

This issue has been identified within the `HookTargetStakeDelegator` contract. When the hook is connected after deposits have been made, the reward vault balance is not properly initialized. As a result, any funds that were deposited before the hook connection cannot be accounted for in the rewards system until they are fully withdrawn and redeposited. This creates a situation where users who deposited before the hook connection will not receive rewards for their existing deposits, effectively losing potential rewards until they perform a complete withdrawal and redeposit cycle.

Recommendation

We recommend implementing a migration process that can be triggered by users (only account owners) to properly account for their existing deposits without requiring a full withdrawal and redeposit cycle. This process should:

- Check if the account has already performed migration (using a mapping to track migration status)
- If not migrated, read the account's current balance in eVault
- Compare it with the `rewardVault's` balance and add the difference to the balance of the account's owner
- Mark the account as migrated to prevent double initialization

The migration process should be automatically invoked if an account with an unsynced balance in the EVault and RewardVault is met.

Client's Commentary:

We acknowledge the scenario described in the audit and have considered it during the design of the smart contract. Unfortunately, the recommended solution is not feasible. The vaults on which the `HookTargetStakeDelegator` will be installed are already participating in the Proof of Liquidity (POL) and allow their shares (ETokens) to be staked in the Reward Vaults. The installation of the `HookTargetStakeDelegator` on the vault will initiate a transition period where rewards will cease flowing to the EToken Reward Vaults and will start flowing to the `ERC20ShareRepresentation` Reward Vaults. Depending on how this process is executed, there is a possibility of rewards overlap, which could lead to a situation where part of the users' stake is counted twice - once in the EToken Reward Vaults and once in the `ERC20ShareRepresentation` Reward Vaults. To avoid potential issues arising from this, we prefer that users manually migrate their stake by first withdrawing (which enforces unstaking in the EToken Reward Vaults) and then redepositing (which will automatically delegate stake in the `ERC20ShareRepresentation` Reward Vaults).

M-3	Stuck Rewards Due to Incorrect Stake Delegation		
Severity	Medium	**Status**	Acknowledged

Description

This issue has been identified within the `_delegateStake` function of the `HookTargetStakeDelegator` contract.

Currently, if the account owner is still unknown to the `checkVaultStatus()` call, `delegateStake()` `HookTargetStakeDelegator.sol#L278` in favor of the `account` itself. Any rewards earned by this account `RewardVault.sol#L326-L336` in the future. As a result, the portion of rewards earned between the delegation and the withdrawal from that account gets stuck on the `RewardVault` contract.

Recommendation

We recommend avoiding minting and delegating tokens if the account owner is not yet known at the time of the `checkVaultStatus()` call. They can be accounted for using the one-time migration method proposed in the M-2 recommendation.

Client's Commentary:

We acknowledge the side effect described in the audit and have considered it during the design of the smart contract. Unfortunately, the recommended solution is not feasible. Implementing the recommendation would result in smart contracts potentially being unable to interact with Euler vaults. Typically, smart contracts do not interact with Euler vaults via the EVC (registering the owner) but can perform deposits and withdrawals directly on the vaults. Consequently, for smart contracts, it is likely that the true EVC owner is never known, but it can be safely assumed to be the account on which the given operation is performed (and to which `ERC20ShareRepresentation` tokens are delegate staked). Implementing the recommendation would disrupt smart contract integrations with Euler. The risk of rewards being potentially lost in case they are earned by the real non-owner account is known and accepted.

2.4 Low

L-1	Missing RewardVault Deployment Check During Stake Delegation		
Severity	Low	**Status**	Fixed in 64786662

Description

This issue has been identified within the `_delegateStake` function of the `HookTargetStakeDelegator` contract.
The function directly calls `rewardVault.delegateStake(...)` without first verifying whether the `rewardVault` contract has been successfully deployed. In edge cases such as when deployment is asynchronous or delayed this may lead to unexpected reverts and disrupt protocol execution.

Recommendation

We recommend adding a check to confirm the existence of the `rewardVault` contract (e.g., using `extcodesize`) before attempting to delegate stake. If the contract is not yet deployed, the function should simply skip delegation to maintain protocol continuity.

Client's Commentary:

Fixed: PR-274

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

 https://mixbytes.io/

 https://github.com/mixbytes/audits_public

 hello@mixbytes.io

 https://x.com/mixbytes