# SMART CONTRACTS REVIEW

zokyo

April 28th 2025. | v. 1.0

# Security Audit Score

## PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

# # ZOKYO AUDIT SCORING YIELDNEST

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

## SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: 0 points

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 0 Medium issues: 0 points deducted
- 1 Low issue: 1 resolved = 0 points deducted
- 2 Informational issues: 2 resolved = 0 points deducted

Thus, the score is 100

# TECHNICAL SUMMARY

This document outlines the overall security of the Yieldnest smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Yieldnest smart/s contract codebase for quality, security, and correctness.

## Contract Status


There were 0 critical issue found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Yieldnest team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

# Table of Contents

# AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Yieldnest repository:
Repo: https://github.com/yieldnest/yieldnest-protocol/pull/220/files

The last commit - 00067a9937c7344420f7a8b13dbf2faa450535b3

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- StakingNode.sol
- StakingNodesManager.sol

**During the audit, Zokyo Security ensured that the contract:**

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Yieldnest smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Thorough manual review of the codebase line by line.

# Executive Summary

The StakingNodeManager.sol smart contract is responsible for managing the lifecycle of staking nodes within the system. Its core functionalities include creating and initializing staking nodes, registering validators, and handling the upgrade of contract implementations, among other key operations. This contract serves as the central management layer for staking infrastructure, ensuring that nodes operate within the expected protocol parameters and governance rules.

On the other hand, the StakingNode.sol smart contract facilitates delegation and undelegation of authority to an operator. It also manages the queuing and processing of withdrawals from EigenLayer, ensuring a seamless interaction between stakers and the underlying staking infrastructure. Additionally, it plays a critical role in enforcing delegation policies and handling stake movements efficiently to maintain network integrity and security.

# STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Yieldnest team and the Yieldnest team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

**Critical**

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

**High**

The issue affects the ability of the contract to compile or operate in a significant way.

**Medium**

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

**Low**

The issue has minimal impact on the contract’s ability to operate.

**Informational**

The issue has no impact on the contract’s ability to operate.

# COMPLETE ANALYSIS

## FINDINGS SUMMARY

| # | Title | Risk | Status |
|---|---|---|---|
| 1 | Missing event when initializing a staking node | Low | Resolved |
| 2 | Unused and not needed variable | Informational | Resolved |
| 3 | Incorrect handling of uint256 to int256 conversion leading to potential misrepresentation of balance | Informational | Resolved |

**Missing event when initializing a staking node**

**Description:**
The `StakingNodesManager.sol` smart contract implements an `initializeStakingNode()` function which is used to initialize a staking node with the necessary version-specific initializations.

However, this function does not emit any event when a staking node is initialized:

```
function initializeStakingNode(IStakingNode node, uint256 nodeCount)
virtual internal {
        uint64 initializedVersion = node.getInitializedVersion();
        if (initializedVersion == 0) {
            node.initialize(
                IStakingNode.Init(IStakingNodesManager(address(this)),
nodeCount)
            );

            // Update to the newly upgraded version.
            initializedVersion = node.getInitializedVersion();
            emit NodeInitialized(address(node), initializedVersion);
        }

        if (initializedVersion == 1) {
            node.initializeV2(0);
            initializedVersion = node.getInitializedVersion();
        }

        if (initializedVersion == 2) {
            node.initializeV3();
            initializedVersion = node.getInitializedVersion();
        }

        if (initializedVersion == 3) {
            node.initializeV4();
            initializedVersion = node.getInitializedVersion();
        }
```

```
        // NOTE: For future versions, add additional if clauses
that initialize the node
        // for the next version while keeping the previous
initializers.
    }
```

**Impact:**
The `initializeStakingNode()` function is an internal function called by `createStakingNode()` and `upgradeStakingNodeImplementation()`. These two functions emit different events, but neither of them includes the `initializedVersion` utilized. This means that if any back-end systems or smart contracts are listening to these functions, it will not be easy to determine which initialized version is used.

**Recommendation:**
Create a new even emitted within the `initializeStakingNode()` function which indicates the `initializedVersion` used.

## INFORMATIONAL-1 | RESOLVED

**Unused and not needed variable**

**Description:**
The `completeQueuedWithdrawals()` function within the `StakingNode.sol` smart contract calculates the withdrawal roots and store them in the `withdrawalRoot` variable:

```
bytes32 withdrawalRoot =
delegationManager.calculateWithdrawalRoot(withdrawals[i]);
```

However, this variable is not used afterwards.

**Recommendation:**
Remove the mentioned line.

## INFORMATIONAL-2 | RESOLVED

**Incorrect handling of uint256 to int256 conversion leading to potential misrepresentation of balance**

**Location:** StakingNode.sol

The `getETHBalance()` function converts the `totalETHBalance` (representing the ETH balance) to an int256 value. If the converted value is negative, the function sets it to `0`. This behavior can result in the total balance being reported as `0`, even when the node holds a positive balance.

**Recommendation:**
Remove the uint256→int256 conversion.

| | StakingNode.sol<br>StakingNodesManager.sol |
|---|---|
| Re-entrancy | Pass |
| Access Management Hierarchy | Pass |
| Arithmetic Over/Under Flows | Pass |
| Unexpected Ether | Pass |
| Delegatecall | Pass |
| Default Public Visibility | Pass |
| Hidden Malicious Code | Pass |
| Entropy Illusion (Lack of Randomness) | Pass |
| External Contract Referencing | Pass |
| Short Address/ Parameter Attack | Pass |
| Unchecked CALL<br>Return Values | Pass |
| Race Conditions / Front Running | Pass |
| General Denial Of Service (DOS) | Pass |
| Uninitialized Storage Pointers | Pass |
| Floating Points and Precision | Pass |
| Tx.Origin Authentication | Pass |
| Signatures Replay | Pass |
| Pool Asset Security (backdoors in the underlying ERC-20) | Pass |

We are grateful for the opportunity to work with the Yieldnest team.

**The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.**

Zokyo Security recommends the Yieldnest team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo