# YieldNest

## SMART CONTRACTS REVIEW

zokyo

May 7th 2024 | 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.

# HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 1 High issue: 1 resolved = 0 points deducted
- 2 Medium issues: 1 resolved and 1 acknowledged = - 4 points deducted
- 0 Low issues: 0 points deducted
- 2 Informational issues: 2 resolved = 0 points deducted

Thus, 100 - 4 = 96

# 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 contract/s codebase for quality, security, and correctness.

## Contract Status


There were 0 critical issues 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/95/files

Last commit - be177b7c6200e8e4737ebebd246982e20dd60547

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

for ynETH.sol:
- RewardDistributor.sol
- RewardsReceiver.sol
- StakingNode.sol
- StakingNodesManager.sol

For ynLSD:
- YieldNestOracle.sol
- LSDStakingNode.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 Zokyo team has conducted a security audit of the provided codebase. The submitted contracts for auditing are well-crafted and organized. Detailed findings from the audit process are outlined in the 'Complete Analysis' section.

# 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.

## HIGH-1 | RESOLVED

### Possible Inflationary Attack In ynETH May Allow Attackers To Get An Unfair Amount of Shares

The `ynETH.sol` contract relies on the execution layer receiver and the consensus layer receivers to determine the amount of shares to distribute to the user in the form of `ynETH` tokens. Malicious users could forcibly transfer a certain amount of Eth into the execution layer receiver, trigger `processRewards` (which anybody can call) in the rewards distributor then deposit 1 wei of Ether into `ynETH` to cause the next user to receive an unfair amount of tokens. In addition to this, the initial user may be minted a disproportionate amount of `ynETH` tokens as the first depositor will be minted at a rate of 1:1 indefinitely and subsequent users are minted less (depending on the balance of incoming rewards in which case, significantly less).

**Recommendation:**
It's recommended that deposits in the `ynETH` contract are bootstrapped similarly to the `ynLSD` contract to make the bug considerably more expensive to trigger.

## MEDIUM-1 | RESOLVED

### Deprecated ETH Transfer Method May Prevent Validators From Being Registered

The `ynETH.withdrawETH()` method is used by the `StakingNodesManager` contract in order to withdraw Ether and register a new validator which uses `payable(*).transfer(toAmount)`. The original transfer method uses a fixed stipend of 2,300 gas units which may not be sufficient for some contracts to process the transfer resulting in a revert, preventing the registration of a validator.

**Recommendation:**
It's recommended that a low level `.call()` is used to transfer Ether between contracts and EOAs.

# COMPLETE ANALYSIS

## FINDINGS SUMMARY

| # | Title | Risk | Status |
|---|---|---|---|
| 1 | Possible Inflationary Attack In ynETH May Allow Attackers To Get An Unfair Amount of Shares | High | Resolved |
| 2 | Deprecated ETH Transfer Method May Prevent Validators From Being Registered | Medium | Resolved |
| 3 | Lack Of Checks To Prevent Adding Duplicate Assets in the Initialize() Functions | Medium | Acknowledged |
| 4 | Unused Custom Errors | Informational | Resolved |
| 5 | Multiple Read Operations Against Storage Variables | Informational | Resolved |

### Lack Of Checks To Prevent Adding Duplicate Assets in the Initialize() Functions

There exists no checks to prevent adding duplicate assets when attempting to initialize the `ynLSD` contract. The impact of duplicate assets lies in the `totalAssets()` function which determines the shares distributed to the users via the `deposit()` function. This may result in a larger value than expected.

**Recommendation:**
It's recommended that the `ynLSD` contract checks for its corresponding EigenLayer Strategy contract in order to prove existence when reinitializing or check if the asset exists in the assets array although the latter could be quite gas intensive.
Client comment: The decision is to acknowledge and leave this as is. The initialization is performed by the YieldNest DAO at launch time and presence of duplicates would be assumed to be avoided. If the ynLSD contract has duplicate assets it's immediately obvious at initialization time and is considered forfeit.

**INFORMATIONAL-1** | RESOLVED

### Unused Custom Errors

The following custom errors are not used throughout the codebase:

- StakingNode.sol, *L52-L55*
- StakingNodesManager.sol, *L42, 45*
- LSDStakingNode.sol, *L35-L36*

**Recommendation:**

It's recommended that these custom errors are removed.

**INFORMATIONAL-2** | RESOLVED

### Multiple Read Operations Against Storage Variables

The `withdrawETH()` function of the `ynETH` contract, the `totalDepositedInPool` variable is read multiple times directly from storage. The `SLOAD` opcode (cold read) will cost 2,100 gas units and 100 units of gas every read after that (warm read) which can add up to be a considerable amount.

**Recommendation:**

It's recommended that storage variables across the code base are first cached to memory which leverages the `MLOAD` instruction and will only code a minimum of 3 gas units for each read operation.

| | RewardDistributor.sol<br>RewardsReceiver.sol<br>StakingNode.sol<br>StakingNodesManager.sol<br>YieldNestOracle.sol<br>LSDStakingNode.sol |
|---|---|
| Reentrance | 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