SMART CONTRACTS REVIEW

zokyo

December 12th 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.

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
- 1 High issue: 1 resolved = 0 points deducted
- 1 Medium issue: 0 points deducted
- 2 Low issues: 1 resolved and 1 acknowledged = - 2 points deducted
- 2 Informational issues: 1 resolved and 1 acknowledged = 0 points deducted

Thus, 100 - 2 = 98

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-vault/tree/feature/x-vault

Last fix - https://github.com/yieldnest/yieldnest-vault/pull/36/files

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

- src/BaseVault.sol
- src/module/Guard.sol
- src/module/Provider.sol
- src/Vault.sol
- src/ynETHxVault.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 smart contracts implements a vault system that manages assets and their associated shares. It enables users to deposit ERC20 tokens or ETH and receive shares that represent their proportional ownership in the vault. The contract includes functionality for withdrawing assets by burning corresponding shares, ensuring that withdrawals align with the user's share of the vault's total assets. It tracks total assets, individual balances, and share allocations to maintain accurate accounting. The design also supports interactions with external strategies for asset management, providing a framework for handling deposits and withdrawals within a decentralized ecosystem.

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	Incorrect Asset Decimals Retrieval	High	Resolved
2	Shadowed decimals Variable in initialize Functions	Low	Resolved
3	Lack of checking if the native coins balance of the contract is sufficient to call the targets	Low	Acknowledged
4	Floating Pragma	Informational	Acknowledged
5	Unused import	Informational	Resolved

HIGH-1 | RESOLVED

Incorrect Asset Decimals Retrieval

The function `addAsset` in BaseVault uses the `decimals_` parameter to set the number of decimals for a newly added asset. However, this value is directly passed by the caller and is not validated against the actual decimals of the ERC20 token. This introduces a risk of misconfiguration, leading to incorrect calculations.

Recommendation:

Retrieve the decimals from the asset contract by calling the `decimals()` function on the ERC20 token contract. This ensures the accuracy of the decimals used in the vault's calculations.

LOW-1 | RESOLVED

Shadowed `decimals` Variable in `initialize` Functions

The `initialize` function includes the `decimals` parameter, which is assigned to the `vaultStorage.decimals` storage variable. However, a `decimals` storage variable exists in all contracts within the current scope, leading to variable shadowing. Although the code functions correctly, this practice can be confusing and error-prone, particularly during debugging or future modifications.

Recommendation:
Rename the `decimals` storage variable in the affected contracts to avoid shadowing. Use a more descriptive name such as `vaultDecimals` or `strategyDecimals`.

LOW-2 | ACKNOWLEDGED

Lack of checking if the native coins balance of the contract is sufficient to call the targets

Location: BaseVault.sol

The processor() function calls the target with sending ether if the value parameter passed is set.

(bool success, bytes memory returnData_) = targets[i].call{value: values[i]}(data[i]);

However, the processor() function is not set payable so the contract could not have enough ether to send when the function is called.

Recommendation:
Make the processor() function payable or add a check if the contract has enough balance to call the targets.

Client's Comment: We don't consider this necessary as the function simply reverts regardless if that happens. It's the same as not having enough approve allowance on an asset.

INFORMATIONAL-1 | ACKNOWLEDGED

Floating Pragma

The contracts use a floating pragma version (`^0.8.24`). Contracts should be deployed using the same compiler version and settings as were used during development and testing. Locking the pragma version helps ensure that contracts are not inadvertently deployed with a different compiler version.

Recommendation:
Consider locking the pragma version to a specific, tested version to ensure consistent compilation and behavior of the smart contract.

Client's Comment: We save broadcast files on every deployment and commit them to the git history and as such we're able to keep clear track of deployment history and versions used. We use the floating pragma to get the latest version and available and deal with any version incompatibilities easier.

INFORMATIONAL-2 | RESOLVED

Unused import

Location: Provider.sol

The Provider contract imports the IChainlinkAggregator interface but it's not used in the contract.

	src/BaseVault.sol src/module/Guard.sol src/module/Provider.sol src/Vault.sol src/ynETHxVault.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 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