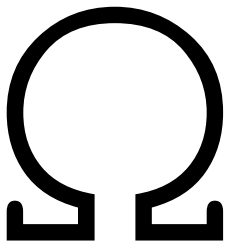
YieldNest PooledDeposits Vault

Final Audit Report

April 25, 2024



Team Omega

Teamomega.eth.limo

Summary

Yieldnest has asked Team Omega to audit the contracts that define the behavior of the `PooledDepositsVault.sol` contract.

We found **no significant issues**. We did however include 3 issues that we classified as "low" and "info" - we believe the code would improve if these issues were addressed.

The issues were subsequently resolved. We reviewed the fixes and commented on the issues below.

Severity	Number of issues	Number of resolved issues
High	0	0
Medium	0	0
Low	2	0
Info	1	0

Scope of the Audit

The scope of the audit is a single file, `PooledDepositVaults.sol`, developed in the following repository

`https://github.com/yieldnest/yieldnest-protocol`

The audit report was based on the following commit:

`8d27c7b37c5b522878c827a3077c578b2057f56d`

The issues were subsequently addressed in commit

`f640856a21b5481289e29140a639b5f570b6fa40`

Methods Used

The contracts were compiled, deployed, and tested in a test environment.

Code Review

We manually inspected the source code to identify potential security flaws.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. No high severity issues were identified with the automated processes. Some low severity issues were found, and we have included them below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.

Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

G1. Pin versions of solidity dependencies [low] [resolved]

In `.gitmodules`, a number of solidity dependencies are specified:

```
[submodule "lib/openzeppelin-contracts-upgradeable"]
      path = lib/openzeppelin-contracts-upgradeable
      url = https://github.com/openzeppelin/openzeppelin-contracts-upgradeable
[submodule "lib/safe-smart-account"]
      path = lib/safe-smart-account
      url = https://github.com/safe-global/safe-smart-account
[submodule "lib/openzeppelin-contracts"]
      path = lib/openzeppelin-contracts
      url = https://github.com/openzeppelin/openzeppelin-contracts
[submodule "lib/eigenlayer-contracts"]
      path = lib/eigenlayer-contracts
      url = https://github.com/layr-labs/eigenlayer-contracts
```

These dependencies do not specify a specific commit or release. This can lead to unexpected problems when a new version of OpenZeppelin or Eigenlayer is merged on github, and other developers (or the continuous integration process, or yourself at a later date) will recompile the contracts with this new version (for example to verify the compiled code on etherscan).
Recommendation: Specify fixed versions of smart contract dependencies instead of ranges, so that the solidity code can be verified and there is no ambiguity about the actual code you are deploying or already have deployed.
Severity: Low
Resolution: The issue was resolved as recommended.

PooledDepositsVault.sol

P1. Centralization risks [low] [acknowledged]

There are two roles that can at any moment remove all funds from the contract or make it unusable.

1. As the contract will be deployed as an upgradeable proxy, and the owner of the proxy can change the implementation contract.
2. The owner of the `PooledDepositsVault` contract can take all funds from the contract by setting the `ynETH` variable to a contract of their choice.

This can be problematic for two reasons. The first is that these two accounts provide an attack surface that, if compromised, can lead to a loss of all funds held in the contract. The second is that users of the contract must trust the account owners to know that their funds are safe: the contract is easily "rugpullable", which may be a reason for users to not deposit their funds.

Recommendation: These risks are to a certain extent unavoidable - the proxy must be managed, and the targeted `ynETH` contract will not be deployed yet when the `PooledDepositsVault` contract is deployed.
However, such risks can be mitigated by having these two roles managed by a multisig. You can also consider managing the upgrade process and the setting of the `ynETH` contract through a timelock, and implement a way for users to opt out (for example, by implementing a withdraw function that allows users to withdraw their ETH). In that way, users can monitor queued transactions and remove their funds if the owner accounts are compromised.
Thirdly, you should also consider whether the upgradeability of such a simple contract is really necessary.
Severity: Low
Resolution: The team has acknowledged the issue, and adds that the Proxy Admin and the Owner of the PooledDepositVault.sol will be 3/5 multisigs operated by DAO members, and that the contracts will be in use for only a limited time.

P2. Declare functions as external instead of public when possible [info] [resolved]

The `initialize` and `setYnETH` functions are marked as `public`, but are not used within the contract and can be declared `external`.
Recommendation: Mark `initialize` and `setYnETH` as `external`.
Severity: Info
Resolution: The issue was resolved as recommended.