Nucleus Security Review

Pashov Audit Group

Conducted by: btk, ubermensch, MrPotatoMagic

December 30th 2024 - December 31th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **Ion-Protocol/nucleus-boring-vault** and **Ion-Protocol/nucleus-queues** repositories was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Nucleus

Ion is a lending built for staked and restaked assets. Borrowers can collateralize their yield-bearing staking assets to borrow WETH, and lenders can gain exposure to the boosted staking yield generated by borrower collateral. The scope for this audit consisted of two separate parts - one for integrating Hyperlane for a crosschain deposit contract, and one for facilitating solver based withdrawals akin to CoW protocol.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hashes:

- 0b5dceb83d12cb146506ee155bb48074fa58c8d8
- 0b3e905fdd5a9eba95eb892c93883428474c5d17)

fixes review commit hash:

- cc250d23ae5b68645a960b55831556489fb46f4b

Scope

The following smart contracts were in scope of the audit:

- `NestAtomicQueue`
- `NestTeller`

7. Executive Summary

Over the course of the security review, btk, ubermensch, MrPotatoMagic engaged with Ion to review Nucleus. In this period of time a total of **2** issues were uncovered.

Protocol Summary

Protocol Name	Nucleus
Repository	https://github.com/Ion-Protocol/nucleus-queues
Date	December 30th 2024 - December 31th 2024
Protocol Type	Lending

Findings Count

Severity	**Amount**
Low	2
Total Findings	**2**

Summary of Findings

ID	Title	Severity	Status
[L-01]	Missing setter functions for deadlinePeriod and pricePercentage	Low	Resolved
[L-02]	Users can grief other users' redeem requests	Low	Acknowledged

8. Findings

8.1. Low Findings

[L-01] Missing setter functions for deadlinePeriod and pricePercentage

The below two variables do not have function setters to allow the owner to update them. Due to this, if the price percentage applied on the atomic price in function requestRedeem() needs to be increased or decreased, it cannot be done.

```
deadlinePeriod = _deadlinePeriod;
        pricePercentage = _pricePercentage;
```

[L-02] Users can grief other users' redeem requests

Users can request a redeem of their shares in a vault at a specific rate determined by the accountant contract through the function `NestAtomicQueue::requestRedeem`.

A solver can then process multiple such requests using the `AtomicQueueUCP::solve` function:

```
function solve(
    ERC20 offer,
    ERC20 want,
    address[] calldata users,
    bytes calldata runData,
    address solver,
    uint256 clearingPrice
)
    external
    nonReentrant
{
    if
      (!isApprovedSolveCaller[msg.sender]) revert AtomicQueue__UnapprovedSolveCaller(m
    uint8 offerDecimals = offer.decimals();
    uint256 assetsToOffer = _handleFirstLoop
      (offer, want, users, clearingPrice, solver);
    uint256 assetsForWant = _calculateAssetAmount
      (assetsToOffer, clearingPrice, offerDecimals);

    IAtomicSolver(solver).finishSolve
      (runData, msg.sender, offer, want, assetsToOffer, assetsForWant);

    _handleSecondLoop(offer, want, users, clearingPrice, solver, offerDecimals);
}
```

The solver provides an array of users for whom to solve the swap. However, if the swap for any user in the array fails, the entire transaction fails, leading to griefing issue.

Failures can occur in multiple ways:

- A user can revoke their allowance for the contract.
- A user can update their request to have an offerAmount of 0.

```
function _handleFirstLoop(
    ERC20 offer,
    ERC20 want,
    address[] calldata users,
    uint256 clearingPrice,
    address solver
)
    internal
    returns (uint256 assetsToOffer)
{
    for (uint256 i = users.length; i > 0;) {
        unchecked {
            --i;
        }

        address user = users[i];
        AtomicRequest memory request = userAtomicRequest[user][offer][want];
        bytes32 key = keccak256(abi.encode(user, offer, want));

        uint256 isInSolve;
        assembly {
            isInSolve := tload(key)
        }

        if (isInSolve == 1) revert AtomicQueue__UserRepeated(user);
        if
          (block.timestamp > request.deadline) revert AtomicQueue__RequestDeadlineExce
=>        if (request.offerAmount == 0) revert AtomicQueue__ZeroOfferAmount
  (user);
        if
          (request.atomicPrice > clearingPrice) revert AtomicQueue__PriceAboveClearing

        assembly {
            tstore(key, 1)
        }

        assetsToOffer += request.offerAmount;
=>        offer.safeTransferFrom(user, solver, request.offerAmount);
    }
}
```

Instead of reverting the entire transaction when a single user's solve fails, modify the logic to skip the failing user and continue processing the remaining users in the array.