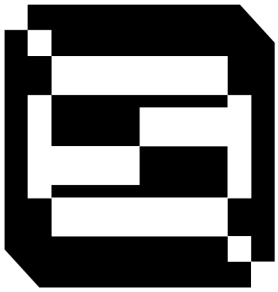

OFFSIDE Labs

Mind Network Restaking

Smart Contract Security Assessment

April 2024

Prepared for:

Mind Network

Prepared by:

Offside Labs

Tim Li

Allen Xu

Siji Feng

Contents

1 About Offside Labs

Offside Labs is a leading security research team, composed of top talented hackers from both academia and industry.

We possess a wide range of expertise in modern software systems, including, but not limited to, *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. We are also at the forefront of innovative areas like *cryptocurrencies* and *blockchain technologies*. Among our notable accomplishments are remote jailbreaks of devices such as the **iPhone** and **PlayStation 4**, and addressing critical vulnerabilities in the **Tron Network**.

Our team actively engages with and contributes to the security community. Having won and also co-organized *DEFCON CTF*, the most famous CTF competition in the Web2 era, we also triumphed in the **Paradigm CTF 2023** within the Web3 space. In addition, our efforts in responsibly disclosing numerous vulnerabilities to leading tech companies, such as *Apple*, *Google*, and *Microsoft*, have protected digital assets valued at over **$300 million**.

In the transition towards Web3, Offside Labs has achieved remarkable success. We have earned over **$9 million** in bug bounties, and **three** of our innovative techniques were recognized among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs

2 Executive Summary

Introduction

Offside Labs completed a security audit of *Mind Network Restaking* smart contracts, starting on April 24, 2024, and concluding on April 24, 2024.

Project Overview

This project consists of Solidity smart contracts that facilitate users in restaking their LRT (Liquid Restaking Tokens) and LST (Liquid Staking Tokens) to the *Mind Network*. The purpose of these contracts is to enhance network security by allowing token holders to actively participate in network operations through restaking.

Audit Scope

The assessment scope contains mainly the smart contracts of the *Restaking Contract* program for the *Mind Network* project.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- mind-restaking-contracts
  - Branch: main
  - Commit Hash: e875ce47012e38f5a20a8ef87f56a1303c28c06f
  - Codebase Link

We listed the files we have audited below:

- mind-restaking-contracts
  - contracts/strategies/Strategy.sol
  - contracts/strategies/IStrategy.sol
  - contracts/tokens/IXERC20.sol

Findings

The security audit revealed:

- 0 critical issue
- 0 high issues
- 0 medium issues
- 3 low issues
- 0 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Redeem Request Manipulation Allows Arbitrary Increase in Redeem Amount Without Extending Lock Period	Low	Fixed
02	Bypassing Limits on Deposit and Redeem Amounts Through Multiple Calls within a Single Transaction	Low	Acknowledged
03	Precision Loss in Asset to Share Conversion Leads to Token Drain in quickWithdraw Function	Low	Fixed

4 Key Findings and Recommendations

4.1 Redeem Request Manipulation Allows Arbitrary Increase in Redeem Amount Without Extending Lock Period

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Rounding Error

Description

In the smart contract `Strategy.sol` function `_updateRedeemLockPeriod`, there is a logic flaw that allows attackers to arbitrarily increase the amount of their redeem request without changing the lock period expiration, provided that the `newRequestAmount` is sufficiently small compared to the `pendingRedeemRequest`.

```
function _updateRedeemLockPeriod(address user, uint256 newRequestAmount)
    private {
...
    } else {
        uint256 newLockPeriod = (newRequestAmount *
            lockPeriod +
            pendingRedeemRequest[user] *
            (pendingRedeemRequestDeadline[user] - block.number)) /
                (newRequestAmount + pendingRedeemRequest[user]);
        pendingRedeemRequest[user] += newRequestAmount;
        pendingRedeemRequestDeadline[user] = block.number +
    newLockPeriod;
    }
}
```

This is due to the way `newLockPeriod` is calculated using integer division, which results in the floor value of the division. Specifically, if the additional `newRequestAmount` is small in comparison to the existing `pendingRedeemReques` amount, and the `lockPeriod` is relatively large, the calculated `newLockPeriod` can end up being equal to the remaining lock period (`pendingRedeemRequestDeadline[user] - block.number`), thus not extending the deadline as expected.

Impact

This vulnerability can be exploited by attackers to significantly increase their redeem request amount while keeping the original lock period deadline intact. This can lead to unexpected states in the smart contract, potentially disrupting the intended redeem mechanisms and leading to financial loss or unfair advantage.

Proof of Concept

Consider the following scenario:

- `newRequestAmountis` 100
- `lockPeriod` is 500
- `pendingRedeemRequestDeadline[user] - block.numbe` is 100
- `pendingRedeemRequest[user]` is 50,000

According to the current implementation, the `newLockPeriod` will still equal 100, allowing the attacker to increment the redeem amount by a small quantity without affecting the final deadline. This issue can be exacerbated through a loop that repeatedly calls `requestRedeem` , enabling the attacker to indefinitely increase the redeem amount.

Recommendation

It is advisable to redesign the withdrawal mechanism altogether based on the requirements.

Mitigation Review Log

Mind Network: **Fixed** by rounding up. Commit *e875ce4*.

4.2 Bypassing Limits on Deposit and Redeem Amounts Through Multiple Calls within a Single Transaction

Severity: Low	Status: Acknowledged
Target: Smart Contract	Category: Logic Error

Description

The smart contract contains functions `deposit` and `requestRedeem` that impose maximum limits on the amounts that can be deposited (`depositAmountMax`) and redeemed (`redeemAmountMax`). However, due to the contract's design, these limits can be easily circumvented by making multiple calls to `deposit` or `requestRedeem` within the same transaction. This oversight allows users to exceed the intended maximum deposit or redeem amounts by splitting the total amount into multiple smaller amounts that each fall below the set limit, thus bypassing the intended restrictions in a single transaction.

```
function deposit(uint256 assetAmount) external whenNotPaused nonReentrant
    {
    if (assetAmount > depositAmountMax) {
        revert ExceededMax();
    }
    _depositFor(_msgSender(), assetAmount, _msgSender());
}
```

Impact

This vulnerability undermines the contract's security and operational integrity by allowing users to inject or withdraw amounts that exceed the designed thresholds. Such actions can lead to unexpected contract behavior, potentially destabilize the contract's economic model, and expose it to risks of liquidity drain or manipulation.

Proof of Concept

A user wishing to deposit or redeem an amount greater than the allowed maximum can do so by splitting the desired total amount into smaller increments that comply with the `depositAmountMax` or `redeemAmountMax` limits. By initiating multiple calls to `deposit` or `requestRedeem` within a single transaction, the user can collectively deposit or redeem an amount that surpasses the intended limit without triggering the `ExceededMax` revert condition.

Recommendation

It is advisable to redesign the withdrawal mechanism altogether based on the requirements.

Mitigation Review Log

Mind Network: Acknowledged. Both `depositAmountMax` and `redeemAmountMax` are designed solely for pausing a single deposition and redemption. It is acceptable to bypass the limit by repeatedly calling these functions.

4.3 Precision Loss in Asset to Share Conversion Leads to Token Drain in quickWithdraw Function

Severity: Low	Status: Fixed
Target: Smart Contract	Category: Rounding Error

Description

In the `quickWithdraw` function, `assetAmount` is converted to `shareAmount` using `Math.Rounding.Floor` for downward rounding. This introduces a precision issue where fewer `shareTokens` are burned for a given `assetAmount`, allowing users to potentially withdraw more `assetTokens` than their `shareTokens` should allow.

```
function quickWithdraw(uint256 assetAmount) external whenNotPaused
    ↪ nonReentrant {
    if (lockPeriod != 0) {
        revert QuickWithdrawalDisabled();
    }
    uint256 shareAmount = _convertToShares(assetAmount,
        ↪ Math.Rounding.Floor);
```

Impact

The rounding error could enable an attacker to exploit the function by repeatedly withdrawing `assetTokens` while minimizing the `shareTokens` burned. This could lead to significant financial discrepancies, undermining the tokenomics and potentially depleting the asset pool more rapidly than anticipated.

Proof of Concept

Consider a scenario where `assetAmount` calculates to a `shareAmount` of 100.5 `shareTokens`. Using `Math.Rounding.Floor`, this is rounded down to 100 `shareTokens`. An attacker can repeatedly perform transactions where they benefit from this rounding error, slowly draining more `assetTokens` than their `shareTokens` cover:

1. The attacker calls `quickWithdraw` with an `assetAmount` leading to a fractional `shareAmount`.
2. The rounding down effectively grants the attacker more `assetTokens` per `shareToken` burned.
3. Over many transactions, this can lead to a substantial loss of `assetTokens` not compensated by an equivalent burn of `shareTokens`.

Recommendation

To rectify this issue, switch the rounding method from `Math.Rounding.Floor` to `Math.Rounding.Ceiling`. This ensures that any fractional `shareToken` requirements are rounded up, thereby securing that every withdrawal of `assetTokens` is fully backed by an appropriate burn of `shareTokens`. This change will prevent the described exploitation mechanism, reinforcing the financial integrity of the system.

Mitigation Review Log

Mind Network: **Fixed** by rounding up. Commit e875ce4.

5 Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as investment advice. While we strive to thoroughly review and analyze the smart contracts in question, we must clarify that our services do not encompass an exhaustive security examination. Our audit aims to identify potential security vulnerabilities to the best of our ability, but it does not serve as a guarantee that the smart contracts are completely free from security risks.

We expressly disclaim any liability for any losses or damages arising from the use of this report or from any security breaches that may occur in the future. We also recommend that our clients engage in multiple independent audits and establish a public bug bounty program as additional measures to bolster the security of their smart contracts.

It is important to note that the scope of our audit is limited to the areas outlined within our engagement and does not include every possible risk or vulnerability. Continuous security practices, including regular audits and monitoring, are essential for maintaining the security of smart contracts over time.

Please note: we are not liable for any security issues stemming from developer errors or misconfigurations at the time of contract deployment; we do not assume responsibility for any centralized governance risks within the project; we are not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By using this report, the client acknowledges the inherent limitations of the audit process and agrees that our firm shall not be held liable for any incidents that may occur subsequent to our engagement.

This report is considered null and void if the report (or any portion thereof) is altered in any manner.