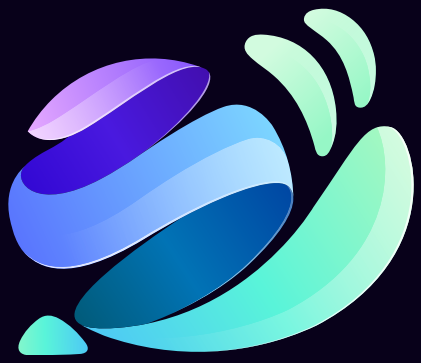


SMART CONTRACTS REVIEW

zokyo

April 10th 2024 | v. 2.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING SNAILBROOK

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
- 0 High issues: 0 points deducted
- 1 Medium issue: 1 acknowledged = -3 points deducted
- 3 Low issues: 3 resolved = 0 points deducted
- 1 Informational issue: 1 resolved = 0 points deducted

Thus, 100 - 3 = 97

TECHNICAL SUMMARY

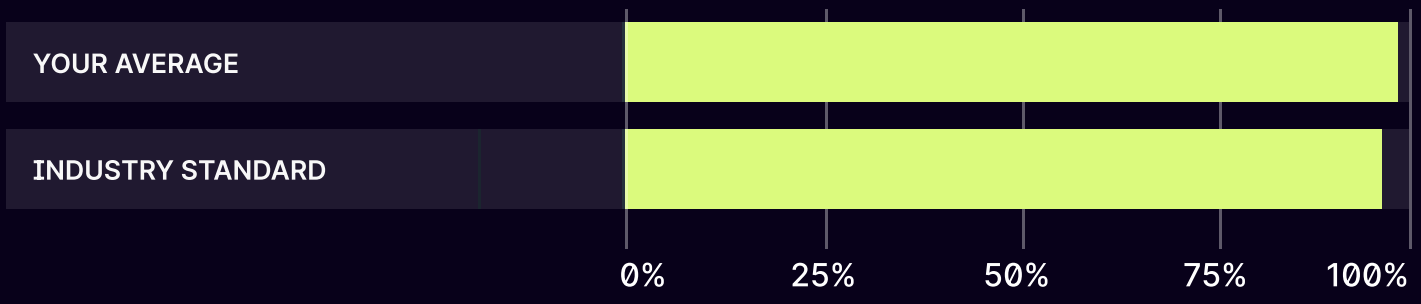
This document outlines the overall security of the SnailBrook smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the SnailBrook smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

Testable Code



98,23% of the code is testable, which is above the industry standard of 95%.

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 SnailBrook 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 SnailBrook repository:
Repo: https://github.com/SnailBrook/SnailBrook-Staking-Contracts

Last commit: 9755fb1a1962a2a416e7d967b2faa2bd6ce6aa06

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

- ./StakingConfigurator.sol
- ./PearlPointsCalculator.sol
- ./StakingManager.sol
- ./StakingRewardsManager.sol
- ./structs/UserStake.sol
- ./structs/StakingInterval.sol
- ./structs/StakingPool.sol
- ./structs/PoolRewards.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 SnailBrook 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. Part of this work includes writing a test suite using the Hardhat testing framework. 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.	03	Testing contract/s logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contract/s by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team has performed a security audit of the provided codebase. The contracts submitted 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 SnailBrook team and the SnailBrook 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	Introduce an emergency withdraw function	Medium	Acknowledged
2	Missing zero address check for the constructor of PearlPointsCalculator & StakingReardsManager	Low	Resolved
3	Renounce Ownership can be called accidentally	Low	Resolved
4	Transfer Ownership is 1-step and can lead to irrecoverable transfers	Low	Resolved
5	The nonReentrant modifier is unnecessary in depositRewards of StakingRewardsManager	Informational	Resolved

MEDIUM-1 | ACKNOWLEDGED

Introduce an emergency withdraw function

The token being used in the system is the SnailBrook Token which is a token with a blacklist , i.e. an address could be blacklisted if malicious activities are found associated with that address. Therefore , it is possible that a user deposits some tokens into the StakingManager contract and due to some suspicious activities that user gets blacklisted , that user would not be able to call withdraw since the token transfer would revert and those funds would be stuck in the contract forever . Instead have an emergency withdraw function where the owner can rescue these stuck tokens.

Recommendation:

Have an emergency withdraw function where the owner can rescue these stuck tokens.It must be ensured that the address calling this function is trusted and is ideally a multisig with a timelock since it will introduce centralisation risks.

Client comment: We explained our vision on this matter, referring to the fact that we did not want to include "god mode" in this contract. Our team believes that having superpowers for the owner over the contract is worse than when the owner only has the rights that are necessary. The user is more protected if the owner of the contract cannot manipulate their staking under any circumstances.

LOW-1 | RESOLVED

Missing zero address check for the constructor of PearlPointsCalculator & StakingReardsManager

In the contract **PearlPointsCalculator.sol**, there is missing zero address check for **stakingManager** in **constructor**. Given that stakingManager is immutable, if it is accidentally set to zero address, it cannot be changed again.

Similarly, in **StakingRewardsManager.sol**, there is missing zero address check for **token & configurator** in the **constructor**. This can lead to token and configurator accidentally set to zero address as well, and cannot be set again.

Recommendation:

It is advised to add a zero address check for

1. **stakingManager** in the constructor() of **PearlPointsCalculator.sol**
2. **token & configurator** in the constructor() of **StakingRewardsManager.sol**

LOW-2 | RESOLVED

Renounce Ownership can be called accidentally

In **PearlPointsCalculator.sol & StakingConfigurator.sol:** The renounceOwnership function can be called accidentally by the admin, leading to immediate renouncement of ownership to a zero address after which any onlyOwner functions will not be callable which can be risky.

Recommendation:
It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed.

Alternatively, the Renounce Ownership functionality can also be disabled by overriding it.

LOW-3 | RESOLVED

Transfer Ownership is 1-step and can lead to irrecoverable transfers

In **PearlPointsCalculator.sol & StakingConfigurator.sol**: The transferOwnership() function in contract allows the current admin to transfer his privileges to another address. However, inside transferOwnership() , the newOwner is directly stored into the storage owner, after validating the newOwner is a non-zero address, and immediately overwrites the current owner.

This can lead to cases where the admin has transferred ownership to an incorrect address and wants to revoke the transfer of ownership or in the cases where the current admin comes to know that the new admin has lost access to his account.

Recommendation:
It is advised to make ownership transfer a two-step process. Or use Openzeppelin's Ownable2Step instead.

Refer- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol

INFORMATIONAL-1 | RESOLVED

The nonReentrant modifier is unnecessary in depositRewards of StakingRewardsManager

The nonReentrant modifier is unnecessary in depositRewards() function of StakingRewardsManager.sol. This is because there is no practical scenario of a reentrancy vulnerability in depositRewards() function. In addition to that the function is already following the Checks-Effects-Interactions pattern

Recommendation:
The nonReentrant modifier can be safely removed from **depositRewards()** function in order to save gas.

	./StakingConfigurator.sol **./PearlPointsCalculator.sol** **./StakingManager.sol** **./StakingRewardsManager.sol** **./structs/UserStake.sol** **./structs/StakingInterval.sol** **./structs/StakingPool.sol** **./structs/PoolRewards.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 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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting SnailBrook in verifying the correctness of their contract/s code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the SnailBrook contract/s requirements for details about issuance amounts and how the system handles these.

PearlPointsCalculator

setPoolPearlMultiplier

✓ Should set the multiplier successfully for valid values
✓ Should fail when non-owner tries to set a multiplier (65ms)
✓ Should fail when setting a multiplier for an already initialized pool
✓ Should fail when setting a multiplier less than 100
✓ Should set the multiplier successfully to the maximum allowed value
✓ Should set the multiplier successfully for multiple pools (40ms)
✓ Should allow setting a multiplier for multiple pools, including non-existent ones
✓ Should set the multiplier successfully to the minimum valid value

getPoolPearlMultiplier

✓ Should return the correct multipliers for set pools and 0 for unset pools

getPearlPointsForStake

✓ Should calculate pearl points correctly for active stakes and return 0 for withdrawn stakes (38ms)

getTotalPearlPointsForStaker

✓ Should calculate pearl points correctly for all active stakes (45ms)

StakingConfigurator

Deployment

✓ Should initialize with correct state
✓ Should ensure the contract is deployed by the expected owner
✓ Should ensure that the contract's functions are callable only by the owner where applicable

setStakingPeriod

✓ Should correctly set the staking period
✓ Should revert if the period is already set
✓ Should revert if the finish time is before the start time
✓ Should revert if the start time is in the past
✓ Should revert when called by a non-owner
✓ Should revert if the start time equals the end time
✓ Should revert if trying to set a new staking period before the current one has ended
✓ Should correctly report the staking period status

addStakingPool

✓ Should allow the owner to add a staking pool with valid duration
✓ Should revert if duration is zero
✓ Should revert when called by a non-owner
✓ Should correctly increment the pools count with multiple additions
✓ Should correctly report the duration of each added pool
✓ Should allow adding multiple pools, even with the same duration

Pool Existence and Duration

✓ Should correctly report the existence of existing pools
✓ Should revert when querying duration for a non-existing pool
✓ Should correctly report the duration of existing pools

Pool Existence and Duration with Multiple Pools

✓ Should correctly report the duration of multiple pools
✓ Should correctly report the existence of multiple pools
✓ Should revert when querying duration for non-existing pools

Comprehensive Scenarios

✓ Should allow adding pools after the staking period has started
✓ Should allow adding pools before the staking period has started
✓ Should allow adding multiple pools before the staking period starts
✓ Should ensure pool duration changes do not affect existing pools

Pool Duration Verification

✓ Should return the correct duration for the first existing pool
✓ Should return the correct duration for the second existing pool
✓ Should correctly report the duration of pools after adding a new pool
✓ Should revert when querying duration for a non-existing pool

StakingManager

Deposit

✓ allows users to deposit tokens into pools and verifies the Deposited event (115ms)
✓ reverts when the staking amount is zero
✓ reverts when the pool does not exist

Withdraw

✓ allows a user to withdraw tokens and claim rewards after the staking period (109ms)
✓ reverts if trying to withdraw from a non-existent stake
✓ reverts if trying to withdraw a stake that has already been withdrawn (62ms)
✓ reverts if trying to withdraw before the staking period has ended (56ms)
✓ correctly updates the total staked amount after a withdrawal (59ms)

Staking Pools Information

✓ reports the total staked amount for pools correctly (81ms)
✓ reports individual user stakes in a specific pool correctly (63ms)
✓ reports zero APY for a pool with no stakes

APY Calculation

✓ accurately calculates the APY for pools (41ms)
✓ returns zero APY for a pool without stakes
✓ updates the APY calculation correctly after additional deposits (42ms)

User Stakes Information

✓ correctly returns a user's stakes and updates after additional deposits (43ms)
✓ returns an empty array for a user with no stakes
✓ updates stake status to 'Withdrawn' after a user withdraws their stake (63ms)
✓ should revert if stake does not exist and return zero if the user has no stakes

Rewards calculation for user stakes

✓ should return the correct rewards for a stake
✓ should return the correct total rewards for all stakes (85ms)

StakingManager

Program rewards distribution

✓ should distribute correct rewards between users (459ms)

Staking rewards distribution

Top up balance and approve spending for 100 users. Place deposits.
First 10 users deposit twice in the middle of the staking period.
Half of the users withdraw their stakes.
Check rewards for the remaining users and their deposits.
Remaining users withdraw their stakes.
Expect each user balance to be greater than initial balance.
Ensure no more active stakes in the pool.
Ensure all rewards have been distributed.
✓ should correctly maintain hundreds of stakes (62472ms)

Shall return correct rewards for withdrawn stake

✓ should return correct rewards for withdrawn stake (80ms)

StakingRewardsManager

Deployment

✓ initializes with correct token and roles
✓ starts without any pools having rewards

Access Control

Role Management

✓ should allow the admin to grant and revoke roles (81ms)
✓ should prevent non-admins from modifying roles (55ms)

Function Access

✓ should enforce access control on sensitive functions (72ms)

Deposit and Claim

✓ should handle deposit and claim operations correctly (99ms)
✓ should revert on invalid operations (112ms)
✓ should update the claimed amount correctly after multiple claims (69ms)

✓ should enforce pool reward deposit rules (65ms)

Claim Rewards Decreases Pool Amount

✓ should correctly decrease pool rewards amount after a claim (93ms)

75 passing (1m)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	%UNCOVE-RED LINES
PearlPointsCalculator.sol	100	90	100	100	
StakingConfigurator.sol	100	100	100	100	
StakingManager.sol	97.37	85	100	97.92	
StakingRewardsManager.sol	100	86.36	100	100	
All Files	**98.23**	**88.37**	**100**	**98.59**	**180,181**

We are grateful for the opportunity to work with the SnailBrook 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 SnailBrook team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo