SMART CONTRACTS REVIEW

zokyo

October 15th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passed a security audit.

ZOKYO AUDIT SCORING IKIGAI STUDIOS LIMITED

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
- 3 High issues: 3 resolved = 0 points deducted
- 3 Medium issues: 3 resolved = 0 points deducted
- 1 Low issue: 1 resolved = 0 points deducted
- 2 Informational issues: 2 resolved = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

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

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

Contract Status

There were 0 critical issues 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 Ikigai Studios Limited 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 5

Executive Summary 7

Structure and Organization of the Document 8

Complete Analysis 9

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Ikigai Studios Limited repository:

Repo: https://github.com/ReservedSnow673/DMN-Staking-Contract
Last commit - 84e7afa8b30f5bd9074397bb763ac232c91c1a5e

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

- Staking.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 Ikigai Studios Limited 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 protocol implements a rank-based staking system for the $DMN ERC20 token, designed to reward long-term commitment through progressive rank accumulation. At its core, the contract enforces a minimum stake threshold of 10,000 $DMN and tracks user progression through nine ranks (0–8), with each rank earned after seven consecutive days of staking. Users should be capable of adding to their staked balance at any time without disrupting rank progression. The system calculates a composite "Daemon Power" metric derived from both staked amount and achieved rank, serving as the basis for protocol benefits such as airdrops and in-game privileges.

The unstaking mechanism incorporates a multi-phase penalty structure to discourage premature withdrawals. Initiating an unstake triggers an immediate one-rank reduction and begins a 14-day unlock period, with additional restrictions imposed if the unstake amount exceeds 25% of the user's balance—specifically, rank growth is paused until the unlock completes or is cancelled. Upon completing the unlock period, users face tiered rank penalties at withdrawal: stakes of 25% or less incur no additional penalty beyond the initial reduction, withdrawals between 25–50% result in rank being halved (minimum reduction of two ranks), and withdrawals of 50% or more reset rank to zero.

Nonetheless the contract provides a grace restake mechanism spanning seven days from unlock completion, during which users can restore their pre-initiation rank (minus the one-rank penalty) by restaking at least the amount they unstaked. The system exposes comprehensive read-only interfaces for off-chain integrations such as leaderboards and user dashboard.

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 Ikigai Studios Limited team and the Ikigai Studios Limited 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	New unstake can never be started after a withdrawal	High	Resolved
2	Users receive all the missed ranks if they fulfill rank criteria in future	High	Resolved
3	TGE bonus can downgrade an already higher rank	High	Resolved
4	Unsafe Downcasting in _updateRank() May Cause Loss of Rewards	Medium	Resolved
5	Rank can still be advanced while the contract is paused	Medium	Resolved
6	Purpose of restakeInGrace() undermined due to missing reassignment of rankBeforeUnstake	Medium	Resolved
7	No RankUpdated event when rank is restored on cancel	Low	Resolved
8	Lack of Event Emission in setTgeBonus Function	Informational	Resolved
9	Floating Solidity Version Pragma	Informational	Resolved

New unstake can never be started after a withdrawal

Description:
After withdraw(), unstakeInfo.amount is set to `0`, but unstakeInfo.initiatedAt stays non-zero which was assigned during initiateUnstake().
Yet initiateUnstake() requires no unstake in progress:

```
// withdraw()
staker.unstakeInfo.amount = 0; // initiatedAt not cleared

// initiateUnstake()
require(staker.unstakeInfo.initiatedAt == 0, "Unstake already in
progress");
```

This means that if a user partially withdraws his stake then since unstakeInfo.initiatedAt is non-zero he can't make a second withdrawal.If the user does not use restakeInGrace() then the user's funds remaining after the partial withdraw are stuck.

Impact:
User can not initiate any future unstake.

Recommendation:

Ensure the unstake state is fully cleared once the withdrawal flow is finished (or allow re-initiation after grace is over).

Users receive all the missed ranks if they fulfill rank criteria in future

Description:
When rank progression is supposed to “pause,” i.e. _updateRank() just returns without moving the time anchor:

```
if (staker.stakedAmount < MINIMUM_STAKE_FOR_RANK) {
    return; // lastRankUpdateTime not advanced
}
if (staker.unstakeInfo.initiatedAt > 0) {
    uint256 p = (staker.unstakeInfo.amount * 100) / staker.stakedAmount;
    if (p > 25) {
        return; // lastRankUpdateTime not advanced
    }}
```

Later if the user's staked amount crosses MINIMUM_STAKE_FOR_RANK or in a scenario where user unstaked more than 25% of his stake then withdraws or cancels right before the unstake delay then timeStaked = block.timestamp - lastRankUpdateTime and the user gets awarded all missed ranks.

Impact:
Users can open a >25% unstake to “pause,” wait weeks, cancel, and still receive full rank catch-up.

Recommendation:
When pausing/returning in rank update logic, also move the accrual timestamp so paused time does not accrue later.

TGE bonus can downgrade an already higher rank

Description:
On first TGE-eligible stake, rank is forcibly set to 2:

```
if (block.timestamp < tgeBonusEndTime && !hasClaimedTgeBonus[msg.sender]
&& _amount >= MINIMUM_STAKE_FOR_RANK) {
    staker.stakeRank = 2;
    ...
}
```

Though this may seem safe, there can be a scenario where a user staked and was promoted to rank 3 (for example) and only after this tgeBonusEndTime was set by the owner, therefore when the user stakes now his rank will be assigned to 2 incorrectly.

Impact:
A user with rank >2 can be pushed down to 2 simply by staking during TGE, provided the tgeBonusEndTime was initialized after the user initial stake.

Recommendation:
Consider only updating the rank to 2 if the current rank of the user is below 2.

Unsafe Downcasting in _updateRank() May Cause Loss of Rewards

Within the `_updateRank()` function, the contract performs a downcast from uint256 to uint8 when calculating newRank:

```
uint256 timeStaked = block.timestamp - staker.lastRankUpdateTime;
uint256 ranksToGain = timeStaked / RANK_PROGRESSION_PERIOD;

if (ranksToGain > 0) {
    uint8 newRank = staker.stakeRank + uint8(ranksToGain);
    ...
}
```

This downcasting can lead to overflow or truncation if `ranksToGain` exceeds the maximum value storable in a uint8 (255). As a result, users who have staked for long periods (e.g., ~ 4 years or more) may experience unexpected behavior such as loss of accrued rank or rewards.

It is worth noting that in another part of the code, the client handles this situation more safely:

```
uint256 timeStaked = block.timestamp - staker.lastRankUpdateTime;
uint256 ranksToGain = timeStaked / RANK_PROGRESSION_PERIOD;

uint256 targetRank = uint256(staker.stakeRank) + ranksToGain;
```

This approach avoids immediate downcasting until after applying a maximum ceiling to ensure safe conversion.

Impact:
The unsafe downcasting may result in loss of stake rewards

Recommendation:
Adopt a consistent, safer pattern for handling rank calculations by keeping operations in uint256 until after bounds checking or ceiling application which client applies already in another part of the codebase.

Rank can still be advanced while the contract is paused

Description:
updateUserRank() lacks whenNotPaused modifier:

```
function updateUserRank() external {
    _updateRank(msg.sende
```

This means that even when the contract is paused, users can call this to advance their ranks further since the rank is dependent on block.timestamp - staker.lastRankUpdateTime.

Impact:
Users can progress rank while the protocol is “paused”.

Recommendation:
Add a whenNotPaused modifier on the updateUserRank() function.

Purpose of `restakeInGrace()` undermined due to missing reassignment of `rankBeforeUnstake`

During the implementation of fixes related to withdrawals, a serious issue was introduced that undermines the purpose of the `restakeInGrace()` feature. The code deletes the entire `staker.unstakeInfo` struct before reassigning only select attributes. However, one important attribute `rankBeforeUnstake` is not restored after deletion.

```
delete staker.unstakeInfo;
staker.unstakeInfo.gracePeriodEndsAt = gracePeriodEnd;
staker.unstakeInfo.originalUnstakeAmount = originalAmount;
// rankBeforeUnstake is not reassigned
```

Since `rankBeforeUnstake` is used to determine the user's rank restoration during the grace restake process, its omission effectively renders the mechanism non-functional.

Impact
Failure to reassign `rankBeforeUnstake` breaks the core logic of the restakeInGrace feature. Users attempting to restake within the grace period will lose their previous rank information. Which in turn might cause user grief and undermine the trustiness of the protocol.

Recommendation:

Ensure that `rankBeforeUnstake` is preserved before deleting the unstakeInfo struct and reassigned afterward. Alternatively, consider selectively resetting only the necessary fields instead of deleting the entire struct to avoid unintentionally removing critical data.

LOW-1 | RESOLVED

No RankUpdated event when rank is restored on cancel

Description:
cancelUnstake() restores the pre-penalty rank but does not emit RankUpdated event:

```
// cancelUnstake()
staker.stakeRank = staker.unstakeInfo.rankBeforeUnstake;
// no RankUpdated event here
```

Recommendation:
Emit RankUpdated whenever rank changes, including on cancel.

INFORMATIONAL-1 | RESOLVED

Lack of Event Emission in setTgeBonus Function

The setTgeBonus function, which is restricted to the contract owner via the `onlyOwner` modifier, updates a critical parameter (`tgeBonusEndTime`) without emitting an event. Event emissions are a best practice in Solidity as they provide transparency and an immutable on-chain log of significant state changes.

```
function setTgeBonus(uint256 _duration) external onlyOwner {
     tgeBonusEndTime = block.timestamp + _duration;
 }
```

Impact
This makes it difficult for users, third-party tools, or auditors to reliably detect updates to `tgeBonusEndTime`, potentially leading to misunderstandings. Additionally, this lack of transparency can reduce trust in the protocol.

Recommendation:
Introduce an event that records changes to `tgeBonusEndTime` whenever `setTgeBonus` is called.

Floating Solidity Version Pragma

The contract uses a floating Solidity version pragma:

```
pragma solidity ^0.8.20;
```

This allows the compiler to use any version starting from `0.8.20` up to (but not including) `0.9.0`, which can inadvertently introduce changes from future compiler versions which client did not test or aware about.

Impact
Future compiler updates may introduce behavior changes, optimizations, or security fixes that are incompatible with the current contract logic.

Recommendation:
Use a fixed pragma version (`i.e. pragma solidity 0.8.20;`) to ensure consistent and predictable compilation across environments.

	Staking.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 Ikigai Studios Limited 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 Ikigai Studios Limited team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo