SMART CONTRACTS REVIEW

zokyo

August 11th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts 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
- 0 High issues: 0 points deducted
- 0 Medium issues: 0 points deducted
- 2 Low issues: 2 resolved = 0 points deducted
- 2 Informational issue: 2 acknowledged = 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

The source code of the smart contract was taken from the Ikigai Studios Limited repository:
Repo: https://github.com/ReservedSnow673/daemons-contracts-audit/tree/usdc

Fixes - https://github.com/ReservedSnow673/daemons-contracts-audit/pull/1

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

- ancient8Daemons.sol
- daemons.sol
- paymentMaster.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 DaemonsAncient8 contract is an ERC721-based NFT system with role-based access control (RBAC). It allows minting of NFTs by users with the MINTER_ROLE, while the admin (ADMIN_ROLE) manages the base URI and can lock or unlock transfers of NFTs. The contract also includes functionality for withdrawing ERC20 tokens and Ether. Additionally, it ensures that transfers are blocked when the transfersLocked flag is enabled, providing further control over NFT transfers.

The ONFTDaemons contract builds upon the LayerZero protocol for cross-chain NFT transfers. Like DaemonsAncient8, it employs RBAC for minting and administrative tasks such as withdrawing ERC20 tokens and Ether. The contract includes the same functionality for minting NFTs, managing token URIs, and locking transfers. Furthermore, it supports the withdrawal of both ERC20 tokens and Ether by authorized users, ensuring secure management of contract funds.

The PaymentMaster contract focuses on secure fund withdrawals, supporting both Ether (ETH) and USDC. It defines a WITHDRAWER_ROLE to authorize specific users for withdrawals, with checks to ensure sufficient balance. The contract tracks each user's withdrawal history and balance for both ETH and USDC, emitting events for off-chain tracking. Admins can easily retrieve the contract's balance and manage the withdrawal process, making it an efficient tool for handling payments in a secure and auditable manner.

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	Access Control Boundaries Are Violated Which Creates A Central Point of Failure	Low	Resolved
2	Some Special Tokens Like USDT Can't Be Recovered From The daemon's Contract	Low	Resolved
3	Multiple Instances Where Storage Is Directly Operated On In PaymentMaster Contract	Informational	Acknowledged
4	Use Custom Errors Across The Code Base Instead Of Strings For More Efficient Reverts	Informational	Acknowledged

Access Control Boundaries Are Violated Which Creates A Central Point of Failure

Description:
Each of the contracts has certain roles which use permissions to operate the contract in its own way such as the MINTER_ROLE. These roles are initially assigned to one single address which can create a central point of failure.

Impact:
A central point of failure will render the contract completely compromised should that single address be taken over.

Recommendation:
It's recommended that different addresses are used to manage different roles for each of the contracts. If one address is compromised they will only have limited permissions.

Some Special Tokens Like USDT Can't Be Recovered From The daemon's Contract

Description:
Stuck ERC20 tokens can be withdrawn from the daemon contract and the ancient8Daemon's contract →

```
function withdrawERC20(address tokenAddress) external
onlyRole(ADMIN_ROLE) {
        IERC20 token = IERC20(tokenAddress);
        uint256 balance = token.balanceOf(address(this));
        require(balance > 0, "No tokens to withdraw");
        require(token.transfer(_msgSender(), balance), "Transfer
failed");
    }
```

But if the token is USDT which does not return a boolean on transfer then the withdraw would always revert.

Impact:
Tokens like USDT which do not return bool on transfer can not be withdrawn.

Recommendation:
Use SafeERC20 library by OpenZeppelin instead.

INFORMATIONAL-1 | ACKNOWLEDGED

Multiple Instances Where Storage Is Directly Operated On In PaymentMaster Contract

Description:
The withdrawEth and withdrawUSDC functions are used to withdraw their respective tokens from the contract. These functions make a record by modifying state variables amountWithdrawnETH/USDC and withdrawalCountETH/USDC which are directly operated on.

Recommendation:
Its recommended that these variables are cached to memory, operated on then stored in their respective state variables.

INFORMATIONAL-2 | ACKNOWLEDGED

Use Custom Errors Across The Code Base Instead Of Strings For More Efficient Reverts

Description:
Currently, the contracts use require statements and a string within when checking conditions and reverting while displaying error messages. This may be less efficient because custom errors use fixed encoding which requires less storage and fewer operations such as memory allocation and data manipulation.

Recommendation:
Use custom errors for example:

```
error ThisThingWentWrong(string memory);

if(someCondition) {
 revert ThisThingWentWrong();
}
```

	ancient8Daemons.sol daemons.sol paymentMaster.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