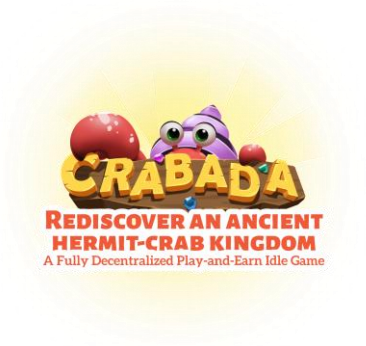
verichains

SECURITY AUDIT OF

CRABADA GAME CONTRACTS



Public Report

Nov 16, 2021

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Nov 16, 2021. We would like to thank the CRABADA for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Crabada game contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Crabada game contracts

A Fully DecentralizedPlay-and-Earn Idle Game

Rediscover the prosperous ancient Crabada Kingdom once ruled by Crustaco, King of the Crabada.

Mine. Loot. Breed. Expand your forces.

Earn CRA tokens by playing and use them to determine the future of the Kingdom!

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of Crabada game contracts. It was conducted on the source code provided by the CRABADA team.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

This report contains sensitive information or information that's not meant to be for the general public. These will be censored out as requested by CRABADA and will be displayed as [redacted].

2. AUDIT RESULT

2.1. Overview

The initial review was conducted on Oct 25, 2021 and a total effort of 4 working days was dedicated to identifying and documenting security issues in the code base of the Crabada game contracts.

The following files were made available in the course of the review:

SHA256 SUM	FILE
f5a5401c8c9a988feb7faeff22ecbb1d7db45be4670685f23fd49bceb12656e5	GameStat.sol
28a6d1c226ba3a65a64ac312ed9a3a9aed0f33dc3a4a67b61637a5d5e2f0bae2	IdleGame.sol
94a227d3e98ebf2e7917450399f6f074ec6a8e582c359c21844087f85fa934fb	IdleGameStorage.sol
aa661f60aaa42a8bbf4a4ca2c1a84c388a89e47a78e22020d258a71da28d675e	IdleGameUpgradable.sol

2.2. Findings

The Crabada game contracts was written in Solidity language, with the required version to be ^0.8.0. The source code was written based on OpenZeppelin's library.

CRABADA fixed the code according to Verichain's draft report

2.2.1. [redacted] - Anyone can call attack using other's team CRITICAL

Function attack doesn't check for attackTeam.owner == msg.sender so anyone can call attack using teams they don't own.

[redacted]

RECOMMENDATION

Add require(attackTeam.owner == msg.sender, "GAME:NOT TEAM OWNER"); check.

[redacted]

UPDATES

- *Nov 15, 2021*: This issue has been acknowledged and fixed by the CRABADA team.

2.2.2. [redacted] - Users can create a team using only one Crabada CRITICAL

Function createTeam let users create a team with 3 Crabada IDs, but it doesn't check those Crabada IDs must not be the same so users can form a team with only 1 Crabada.

[redacted]

RECOMMENDATION

Add require(crabadaId1 != crabadaId2 && crabadaId2 != crabadaId3 && crabadaId3 != crabadaId1, "GAME: CRABADA IDS MUST NOT BE THE SAME"); to createTeam.

[redacted]

UPDATES

- *Nov 15, 2021*: This issue has been acknowledged and fixed by the CRABADA team.

2.2.3. [redacted] - Modifier whenNotPaused on function deposit can be bypassed LOW

Function deposit has modifier whenNotPaused but users can bypass it by directly transferring to the contract because onERC721Received hook doesn't have whenNotPaused modifier.

[redacted]

RECOMMENDATION

Add whenNotPaused() modifier to onERC721Received.

[redacted]

UPDATES

- *Nov 15, 2021*: This issue has been acknowledged by the CRABADA team.

2.2.4. [redacted] - Redundant mustUpdateStat modifier INFORMATIVE

Function attack updates stat of attack team by _updateStat(attackTeamId) so the modifier ustUpdateStat(attackTeamId) is redundant.

[redacted]

RECOMMENDATION

Consider removing redundant modifier _updateStat(attackTeamId) in attack function.

[redacted]

UPDATES

- *Nov 15, 2021*: This issue has been acknowledged and fixed by the CRABADA team.

2.2.5. [redacted] - Gas optimize INFORMATIVE

In _settleGame function, stat variable is using as read only variable so no need to declare a memory variable (wasting gas to copy variable from storage to memory).

[redacted]

RECOMMENDATION

Consider change memory to storage for stat variable for gas saving.

[redacted]

UPDATES

- *Nov 15, 2021*: This issue has been acknowledged and fixed by the CRABADA team.

2.2.6. [redacted] - Typo in _settleGame function INFORMATIVE

There is a typo in variable isCraRewardAvaialbe of _settleGame function.

[redacted]

RECOMMENDATION

Fix the typo.

UPDATES

- *Nov 15, 2021*: This issue has been acknowledged and fixed by the CRABADA team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Nov 15, 2021*	Public Report	Verichains Lab
1.1	*Nov 16, 2021*	Public Report	Verichains Lab

Table 2. Report versions history