Coinsult

Request your audit at coinsult.net

Advanced Manual

Smart Contract Audit

October 6, 2023

Audit requested by

RaceGame

Table of Contents

Audit Summary

Project Name	RaceGame
Website	-
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x63aB3991617F5c849f269b769b588AA5C4317441
Audit Method	Static Analysis, Manual Review
Start date of audit	October 6, 2023

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Note: This audit report is the 4th version, previous audits have spotted several issues which have all been resolved by the developing team.

Audit Scope

Source Code

Coinsult was comissioned by RaceGame to perform an audit based on the following code:

https://bscscan.com/address/0x63aB3991617F5c849f269b769b588AA5C4317441#code

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used Tools

- ✓ Slither: Solidity static analysis framework
- ✓ Remix: IDE Developer Tool
- ✓ CWE: Common Weakness Enumeration
- ✓ SWC: Smart Contract Weakness Classification and Test Cases
- ✓ DEX: Testnet Blockchains

Static Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Failed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	Passed

SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Does not compromise the functionality of the contract in any way	Passed
SWC-130	Does not compromise the functionality of the contract in any way	Passed
SWC-131	Does not compromise the functionality of the contract in any way	Passed
SWC-132	Does not compromise the functionality of the contract in any way	Passed
SWC-133	Does not compromise the functionality of the contract in any way	Passed
SWC-134	Does not compromise the functionality of the contract in any way	Passed
SWC-135	Does not compromise the functionality of the contract in any way	Passed
SWC-136	Will definitely cause problems, this needs to be adjusted	Passed

Note: This audit report is the 4th version, previous audits have spotted several issues which have all been resolved by the developing team. No issues are left to be handled by the team.

Constructor Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract RaceGame {
    event RaceCreated(uint raceId);
    event JoinRace(uint raceId, address joined, uint nftJoined);
    event RaceStarted(uint raceId);
    event RaceFull(uint raceId);
    event RaceComplete(uint raceId, address winner, uint amount);

    address DEAD = 0x000000000000000000000000000000000000dEaD;

    struct Race {
        uint id;
        string eventName;
        uint status; // 0 initialized, 1 complete, 2 finished
        address[] players;
        uint256[] nftIds;
        uint maxPlayers;
        uint entryFee;
        bool started;
```

Disclaimer