CERTIK

# CertiK Audit Report for Zen Sports

# Contents

## Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Zen Sports (the “Company”), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the “Agreement”). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK’s prior written consent.

## About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK’s mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance’s BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable.  For more information: https://certik.io.

## Executive Summary

This report has been prepared for **Zen Sports** to discover issues and vulnerabilities in the source code of their **Smart Contract** as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

## Testing Summary

SECURITY LEVEL

TIER-ONE
VERY HIGH CONFIDENCE
VERIFIED BY CERTIK

Smart Contract Audit
This report has been prepared as a product of the Smart Contract Audit request by Zen Sports.
This audit was conducted to discover issues and vulnerabilities in the source code of the Zen Sports Smart Contract.

| | |
|---|---|
| TYPE | Smart Contract |
| SOURCE CODE | https://tracker.icon.foundation/contract/cx08eb683a53d6f244d538a45f0cc584825e2cb985#code |
| PLATFORM | ICON |
| LANGUAGE | Python |
| REQUEST DATE | Aug 10, 2020 |
| DELIVERY DATE | Aug 19, 2020 |
| METHODS | A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review. |

## Review Notes

### Introduction

CertiK team was contracted by the Zen Sports team to audit the design and implementation of their token smart contract and its compliance with the IIPs it is meant to implement.

The audited source code link is:

- https://tracker.icon.foundation/contract/cx08eb683a53d6f244d538a45f0cc584825e2cb985#code

The goal of this audit was to review the Python implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

## Documentation

The sources of truth regarding the operation of the contracts in scope were minimal although the token fulfilled a simple use case we were able to fully assimilate. To help aid our understanding of the contract's functionality we referred to in-line comments and naming conventions.

## Summary

The codebase of the project is a typical IIP2 implementation.

**Optimization steps** that we pinpointed in the source code only referred to coding standards and inefficiencies. No **minor (or above) flaws** were identified.

The codebase of the project strictly adheres to the standards and interfaces imposed by the ICON open-source libraries and as such its typical IRC-2 functions **can be deemed to be of high security and quality**.

## Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report **to achieve a high standard of code quality and security**.

## Findings

### Exhibit 1

| TITLE | TYPE | SEVERITY | LOCATION |
|---|---|---|---|
| Transfer Function Address Restriction | Coding Style | Informational | Lines 102-121 |

**[INFORMATIONAL] Description:**

Transferring from and to the system address (zero address) should be restricted.

**Recommendations:**

We advise that the team adds an "if" block checking if the sender or the recipient is the system address.

Example:

```
If _from == SYSTEM_SCORE_ADDRESS or _to == SYSTEM_SCORE_ADDRESS:
        revert("Cannot Transfer from and to zero address")
```

## Exhibit 2

| TITLE | TYPE | SEVERITY | LOCATION |
|---|---|---|---|
| Secure Math Operations | Coding Style | Informational | Lines 110, 111 |

**[INFORMATIONAL] Description:**

In general, ensuring the safe execution of mathematical operations is essential and an extra layer of security should be added to the codebase.

**Recommendations:**

We advise that the team adds an assertion after each math operation to validate the result of the mathematical operation.

Example:

```
oldBalanceFrom = self._balances[_from]
self._balances[_from] = self._balances[_from] - _value
assert (oldBalanceFrom == self._balances[_from] + _value), "error message"

oldBalanceTo = self._balances[_to]
self._balances[_to] = self._balances[_to] + _value
assert (oldBalanceTo == self._balances[_to] - _value), "error message"
```