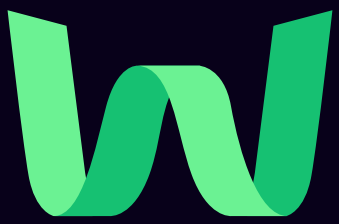

SMART CONTRACTS REVIEW

zokyo

November 10th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING WINS

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
- 0 Low issues: 0 points deducted
- 2 Informational issues: 2 unresolved = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

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

The scope of this audit was to analyze and document the WINS 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 WINS 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 WINS repository:
Repo: https://github.com/TrumanEc/win-sports-token

Last commit -94bd11c0a8272c6a301c06d6890dee80630dae32

Contracts under the scope:

- Scope: WIN.sol

Throughout the review process, care was taken to ensure 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 WINS 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

WIN Investments is a digital economy which guarantees WIN securities, aiming to scale the community, strengthen user loyalty and drive global adoption. Zokyo was tasked with the security audit of the WIN token, which is an ERC20 utility token which is intended to be deployed to the Ethereum mainnet. The goal of the token, whilst it doesn't represent any equity, securities or participation in the company, it's designed to enhance user interaction within the platform.

The win token is a simple implementation that inherits ERC20 and ERC20 permit from the OpenZeppelin contracts which are battle tested and have withstood the test of time. The initial contract owner is transferred 1 billion tokens on deployment which they are responsible for distribution of such tokens. Users are able to do everything that is supported by the ERC20 standard such as transfer tokens, approve tokens to another user, and use ERC20 permits to gaslessly approve certain amounts to other users. Through ownership, according to the documentation, these utility tokens will be backed by real-world legal contracts that ensure legitimacy and proper structure for use within the ecosystem.

Overall the contract is well documented through inline comments and structured accordingly to achieve the goals of a utility token and adheres to security standards without needing to rewrite code to reinvent the wheel. This contract relies on tried and tested dependencies. There are no developer defined sinks or access controls/powers that the initial contract owner has. As a result, this audit only yielded informational severity findings revolving around compiler versioning and slight gas optimisations. Although it is recommended that WIN investments takes care when it comes to distributing the tokens as distribution too slowly can stagnant growth but distribution too quickly can cause significant supply and this see major turbulence in its price.

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 WINS team and the WINS 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	Compiling solidity code using ^ could introduce breaking changes to the code	Informational	Unresolved
2	Using custom errors as opposed to error strings in require statements will use less gas	Informational	Unresolved

Compiling solidity code using ^ could introduce breaking changes to the code

Description

Currently, the WIN ERC20 token is compiled using any compiler version above 0.8.27 but not including 0.9.x. This means that when the code is compiled it will use whichever version is the latest which may introduce unpredictable changes into the code on deployment.

Recommendation:

It's recommended that the latest solidity version is pinned as the contract compiler for example:

```
pragma solidity =0.8.30
```

Using custom errors as opposed to error strings in require statements will use less gas

Description

Within the constructor of the WIN token, there is a require check that the initialOwner is not the zero address and includes an error string. The greater the length of the string the more gas is required to deploy the contract.

Recommendation:

Custom error messages are recommended to optimize the gas usage of this contract during deployment. For example:

```
// ====================== SNIP ======================
    error InvalidInitialOwner();
    /// @notice Constructor for WIN Sports Token contract
    /// @param initialOwner Address that will receive the initial supply
    constructor(
        address initialOwner
    ) ERC20("WIN Sports Token", "WINS") ERC20Permit("WIN Sports Token") {
        if(initialOwner == address(0)) {
            revert InvalidInitialOwner();
        }
// ---------------------- SNIP ----------------------
```

	WIN.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 WINS 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 WINS team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo