QuillAudits

AUDIT REPORT

May , 2025

For

Spor

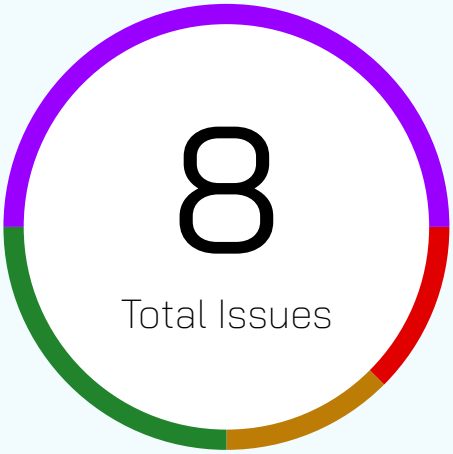
Table of Content

Executive Summary

Project name	Spor
Project URL	https://spore.xyz/
Overview	In the GameManager contract, the openBox function doesn't validate that the nonce belongs to the specific user, allowing potential nonce reuse across different users. In the fulfillRandomWords function, scaling the random value by multiplying by 1e15 could lead to numerical overflow or precision issues when using the randomness for game mechanics.
Audit Scope	The scope of this Audit was to analyze the Spor Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/dtechvision/spor-smart-contracts/tree/master/src
Contracts in Scope	GameManager.sol GameManagerProxy.sol Spor.sol
Branch	Master
Commit Hash	ed423376fd62ea20f0f804aa4128864be2b7d494
Language	Solidity
Blockchain	BSC
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	2nd April 2025 - 15th April 2025

Updated Code Received	25th May 2025
Review 2	25th May 2025 - 27th May 2025
Fixed In	c49d2e0f38b7f8664638a101c8284767d971abff

Number of Issues per Severity



High	1 (12.50%)
Medium	1 (12.50%)
Low	2 (25.00%)
Informational	4 (50.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**1**	**1**	**1**	**1**
Acknowledged	0	0	**1**	**3**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

Security vulnerabilities identified that must be resolved and are currently unresolved.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

High Severity Issues

Possibility of Signature Replay

Resolved

Path

GameManager.sol

Function

openbox

Description

The signature verification in functions like openBox, merge, and mintSpor doesn't include the chainId in the signature data. This enables replay attacks across different networks using the same signature.

Recommendation

Include the chainId in the message being signed

```
bytes32 messageHash = MessageHashUtils.toEthSignedMessageHash(
      keccak256(abi.encode(block.chainid, account, sporCost, id, amount,
data, nonce))
);
```

Medium Severity Issues

Upgradeable Contract lacks gap variable

Resolved

Description

The GameManager contract is upgradeable (inherits from UUPSUpgradeable) but doesn't include a storage gap variable. Without a storage gap, adding new state variables in future contract versions could overwrite existing storage, leading to data corruption or loss.

Recommendation

Add a storage gap variable at the end of the contract to reserve storage slots for future upgrades.

```
// Add this at the end of the contract
uint256[50] private __gap;
```

Low Severity Issues

Use Ownable2Step instead

Resolved

Description

The contracts use direct ownership transfer through OpenZeppelin's Ownable pattern. If ownership is accidentally transferred to an invalid or inaccessible address, control of the contract is permanently lost.

Recommendation

Use OpenZeppelin's Ownable2Step instead, which requires the new owner to accept ownership before the transfer is complete.

Centralization Risk

Acknowledged

Description

The contract contains owner controlled rescue function that have the potential to completely drain the contracts in case of a malicious or compromised owner.

Recommendation

It is recommended to ensure that owner is a multisig wallet.

Informational Severity Issues

Missing event emission for critical state changes

Resolved

Path

Spor.sol

Function

updateGameManager

Description

In both SporItems and Spor contracts, the updateGameManager function doesn't emit events when updating the critical gameManager address.

Recommendation

Implement event emissions for all state-changing functions, particularly those that modify critical addresses or parameters.

Nonce validation is not sequential

Acknowledged

Path

GameManager.sol

Function

openBox

Description

The nonce system uses a boolean mapping to track used nonces (usedNonce[nonce]), which allows for non-sequential nonce usage. This could lead to out-of-order transaction execution and potential security issues.

Recommendation

Implement a sequential nonce system for each user address

Consider declaring MAX_SUPPLY as immutable

Acknowledged

Path

GameManager.sol

Function

openBox

Description

The contract does not declare a MAX_SUPPLY value as immutable, which could enhance gas efficiency and clarity of tokenomics if a maximum token cap is part of the design. Immutable variables are stored directly in the bytecode, making them cheaper to access than storage variables.

Recommendation

Mark this as immutable

Lack of Pause Mechanism

Acknowledged

Path

GameManager.sol

Function

openBox

Description

While the token contracts (Spor and SporItems) have pause functionality, the main GameManager contract lacks a pause mechanism, preventing emergency stops in case of discovered vulnerabilities.

Recommendation

Implement the OpenZeppelin Pausable pattern in GameManager and add modifiers to critical functions

Functional Tests

Some of the tests performed are mentioned below:

- mint function mints amount up to the limit of MAX_SUPPLY
- openbox function uses some robust checks ; but the nonce is not updated in a perfect sequence
- All the functions have robust access control ; but they also pose a centralization risk
- mintSpor function is working as expected
- merge function is working as expected

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Spor. We performed our audit according to the procedure described above.

Issues of High, Medium, Low and Informational severities were found. Spor team fixed some and acknowledged others.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

May , 2025

For

Spor

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com