QuillAudits

Audit Report
November, 2022

For

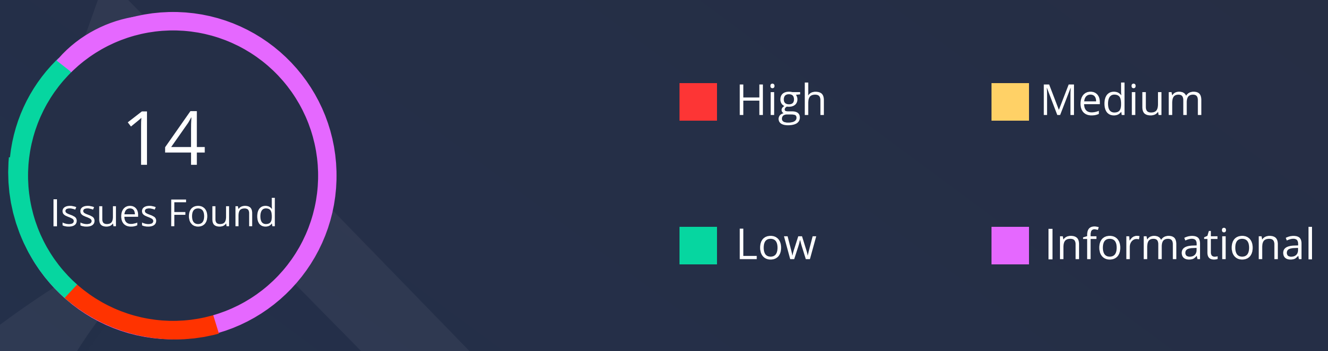
SPORTIQO

Table of Content

Table of Content

Executive Summary

Project Name	Sportiqo
Overview	Sportiqo is a platform where sports fans can trade the performance of their favorite players like trading stocks. It allows fans to apply their sporting IQ and take a view on the performance of a player over the long term such as a season or even their full career.
Timeline	27th Oct 2022 - 9th Nov 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Sportiqo codebase for quality, security, and correctness. *https://bitbucket.org/sricworkspace/smartcontract/src/master/* Commit hash: 902e10d
Fixed In	*https://bitbucket.org/sricworkspace/smartcontract/src/master/* Commit hash: a79f029



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	2	0	2	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	0	3	4

Types of Severities

High

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

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

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

Informational

These are 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

These are the issues identified in the initial audit and have been successfully fixed.

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.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Techniques and Methods

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

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token 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 analysed 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

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 analysed, 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 behaviour 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, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Player Counter

High Severity Issues

A.1 Centralization Risk

Description

The function sendTo() and burn() allows the contract owner to remove all the funds from a user account. This poses a risk for the token holders where their funds can be moved by the contract owner at any time.

Remediation

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions: with reasonable latency for community awareness on privileged operations; Multisig with community-voted 3rd-party independent co-signers; DAO or Governance module increasing transparency and community involvement;

Status

Acknowledged

Medium Severity Issues

No issues found

Low Severity Issues

A2. Floating Pragma

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might negatively introduce bugs that affect the contract system.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to use the 0.8.7 version.

Status

Acknowledged

A.3 Add external modifier instead of public

Description

It is recommended to use external access modifier instead of public for the following functions which are not called from the contract:
activate()
deactivate()
mint()
sendTo()
burn()
makeAdmin()
mintedTokens()

Remediation

As per the solidity security recommendation, the functions should first update the contract states and then interact with external contracts.
Please refer solidity documentation here:
https://docs.soliditylang.org/en/develop/security-considerations.html#use-the-checks-effects-interactions-pattern

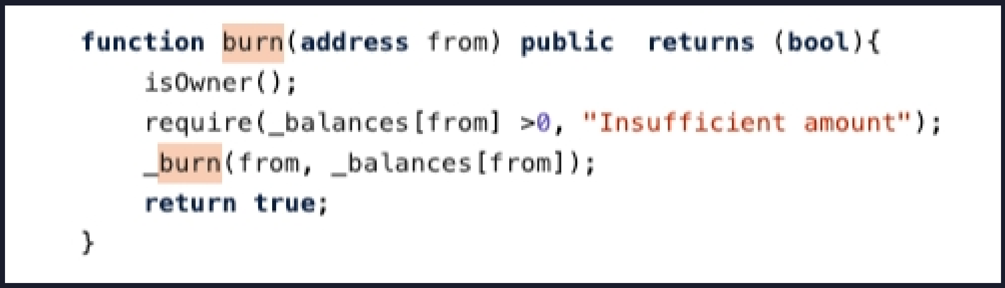
Status

Resolved

A.4 Balance should be checked against amount

Description

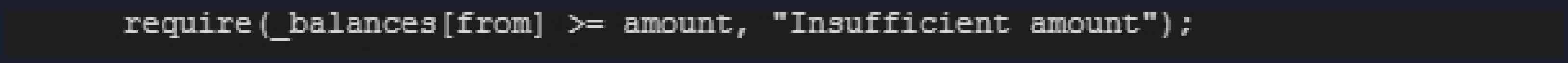
It is recommended to check the amount in the user account instead of non-zero. For solidity versions less than 0.8.0, it can cause integer overflow and allow user to hold unlimited tokens.



```
    function burn(address from) public  returns (bool){
        isOwner();
        require(_balances[from] >0, "Insufficient amount");
        _burn(from, _balances[from]);
        return true;
    }
```

Remediation

Update the require to following:



```
        require(_balances[from] >= amount, "Insufficient amount");
```

Status

Resolved

Informational Issues

A.5: State Variable Default Visibility

Description

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.
address owner
address houseAddr
uint256 _wei

Remediation

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables. Ref: https://swcregistry.io/docs/SWC-108

Status

Resolved

A.6: isOwner can be declared pure.

Description

The function contains a require due to which return is not needed. Also as the function is not using storage, it can be declared pure

Remediation

Make the function pure and remove returns.

Status

Acknowledged

A.7: General Recommendation

Description

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Status

Acknowledged

B. Contract - SPQ

High Severity Issues

B.1 coldWalletAddress() blocks start() function

Description

The function setAddress() allows the contract owner to set one address at a time and once called, no other address can be set. This will block the start() function call.

Remediation

We advise to fix the logic by checking against each address if its value is updated or not.

Status

Resolved

B.2 Centralization Risk

Description

The function sendTo() and burn() allows the contract owner to remove all the funds from a user account. This poses a risk for the token holders where their funds can be moved by the contract owner at any time.

Remediation

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions: with reasonable latency for community awareness on privileged operations; Multisig with community-voted 3rd-party independent co-signers; DAO or Governance module increasing transparency and community involvement;

Status

Acknowledged

Medium Severity Issues

No issues found

Low Severity Issues

B.3 Floating Pragma

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might negatively introduce bugs that affect the contract system.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to use the 0.8.7 version.

Status

Acknowledged

B.4 Add external modifier instead of public

Description

It is recommended to use external access modifier instead of public for the following functions which are not called from the contract:

initialize()
start()
mint()
sendTo()
burn()
enableToken()
makeAdmin()
mintedTokens()
setAddress()
isOwner()

Remediation

As per the solidity security recommendation, the functions should first update the contract states and then interact with external contracts.
Please refer solidity documentation here:
https://docs.soliditylang.org/en/develop/security-considerations.html#use-the-checks-effects-interactions-pattern

Status

Resolved

Informational Issues

B.5: State Variable Default Visibility

Description

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.
owner()
isRun()
isColdWalletAddressSet()
_wei()

Remediation

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables. Ref: *https://swcregistry.io/docs/SWC-108*

Status

Resolved

B.6: isOwner can be declared pure.

Description

The function contains a require due to which return is not needed. Also as the function is not using storage, it can be declared pure.

Remediation

Make the function pure and remove returns

Status

Resolved

Informational Issues

B.7: General Recommendation

Description

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Status

Acknowledged

Functional Testing

- ✓ should be able to deploy and mint the initial token supply.
- ✓ Should be able to mint more tokens.
- ✓ Should be able to transfer ERC-20 to addresses.
- ✓ Should be able to activate and deactivate the contracts.
- ✓ Should be able to burn the tokens from an address.
- ✓ Should revert if burn and mint are not called by owner.
- ✓ Should be able to update admin.
- ✓ Should be able to setAddress.

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 the Sportiqo. We performed our audit according to the procedure described above.

Some issues of High,Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Sportiqo Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Sportiqo Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

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.

700+
Audits Completed

$15B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report November, 2022

For

SPORTIQO

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com