QuillAudits

Audit Report
July, 2023

For

TRUSTX GAMING

Table of Content

Executive Summary 01

Checked Vulnerabilities 03

Techniques and Methods 04

Manual Testing 05

High Severity Issues 05

Medium Severity Issues 06

Low Severity Issues 06

Informational Issues 07

Functional Tests 08

Automated Tests 09

Closing Summary 10

About QuillAudits 11

Executive Summary

Project Name	TRUSTxGAMING
Overview	The main contract is a token contract inheriting the IBEP20 interface, Context, Pausable and Ownable libraries. This contract has properties of a conventional token but is exceptional because these properties have “WhenNotPaused” modifiers that regulate when users can carry out their activities. It also gives the contract owner the privilege to freeze accounts.
Timeline	3 June, 2023 to 5 June, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Language	Solidity
Blockchain	BSC
Scope of Audit	The scope of this audit was to analyze the TRUSTxGAMING codebase for quality, security, and correctness: *https://bscscan.com/address/0x9f46ecf92e7f6ee8c03f393adf04c2e17b8cd0b0#code* Branch: NA Commit: NA
Contracts in Scope	TRUSTxGAMING.Sol
Fixed In	NA

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	1	1	0	2
Resolved Issues	0	0	0	0

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 - TRUSTxGAMING.sol

High Severity Issues

A.1 Contract Owner Can Increase Its Balance When Burning Tokens in Any Account and Causing an Inconsistency with the Total Supply

```
function _burn(address account, uint256 amount) internal {
    require(account != address(0), "BEP20: burn from the zero address");
    require(!frozenAccount[msg.sender]);

    _balances[account] = _balances[account].sub(amount, "BEP20: burn amount exceeds balance");
    _balances[owner()] = _balances[owner()].add(amount);
    _totalSupply = _totalSupply.sub(amount);
    emit Burn(account, amount);
}
```

Description

The burn function can only be called by the contract owner. This implies that the owner can burn tokens from any user account. The issue of increasing the owner's balances arises at the level of the _burn internal function. When the contract owner passes an user account address to the public burn function, it deducts the amount from the balance of the user, increasing the balance of the contract owner but also reduces the total supply value. This will cause an inconsistency with the value of tokens in circulation. There will be more tokens in the contract owner's balance but less tokens noted by the total supply variable.

Remediation

Since the tokens burned are to be redistributed to users, it is recommended to also add the expected amount to be burned with the total supply to maintain a consistency with the value of the total supply.

Status

Acknowledged

Medium Severity Issues

A.2 Centralization Risk

Description

With the use of the Ownable contract inherited in the contract, it gives the contract owner privileges to call some functions. The contract owner can perform actions ranging from burning of tokens from any account, freezing an account which will deny freezed account from carrying out any token activities, pausing, and unpausing the token contract and also withdrawing any kind of token sent into the contract. Users must trust the contract owners to carry out these admin tasks.

Remediation

It is recommended that a trusted multisig address is used to perform admin duties.

Status

Acknowledged

Low Severity Issues

No issues found

Informational Issues

A.3 Double Frozen Account Require Check in the Approve Function

Description

There is a frozenAccount mapping in the contract used to identify accounts that have been frozen. This is however added in some functions to prevent the continuous activity of the frozen account. In the approve and _approve functions, there was a check that appears in both functions. The _approve internal function, this check was made likewise in the public approve function which will cause extra gas cost when the approve function is called by users.

Remediation

To save extra cost incurred when calling the approve function, remove a required check in one of the functions.

Status

Acknowledged

A.4 Insufficient Code Comment and Specification

Description

There were some functions introduced into the token contract and sufficient code comments were not highlighted on them to explain why they were introduced. This also stretches to the smart contract specification provided before the start of the audit.

Remediation

Add more code comment on the additional functions in the contract and also a detailed specification proving data in the contract.

Status

Acknowledged

B. General Recommendation

With the admin task and privileges that the contract owner has, it is expected that users interacting with the token contract trust the contract owner. Hence, there should be a multisig account handling this admin task to mitigate the fear of the centralization risk.

Functional Tests

Some of the tests performed are mentioned below

- Should get the name of the token
- should get the symbol of the token
- should get the symbol of the token
- should get the total supply of the token when deployed
- should get balance of the owner when contract is deployed
- should transfer tokens to other address
- should approve another account to spend token
- should burn the token in any account by the contract owner and increase their balance
- should revert when non-owner is trying to burn
- Should airdrop tokens to an array of addresses with singular amount
- Should successfully multi-send tokens to different addresses and different amount

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.

```
INFO:Detectors:
TRUSTxGAMING.withdraw(address,address,uint256) (contracts/TRUSTxGAMING.sol#667-670) ignores return value by IBEP20(con_address).transfer(to,amount) (contracts/T
RUSTxGAMING.sol#669)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
Contract locking ether found:
        Contract TRUSTxGAMING (contracts/TRUSTxGAMING.sol#405-694) has payable functions:
         - TRUSTxGAMING.withdraw(address,address,uint256) (contracts/TRUSTxGAMING.sol#667-670)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
TRUSTxGAMING.getAllowance(address,address).owner (contracts/TRUSTxGAMING.sol#440) shadows:
        - Ownable.owner() (contracts/TRUSTxGAMING.sol#306-308) (function)
TRUSTxGAMING.allowance(address,address).owner (contracts/TRUSTxGAMING.sol#494) shadows:
        - Ownable.owner() (contracts/TRUSTxGAMING.sol#306-308) (function)
TRUSTxGAMING._approve(address,address,uint256).owner (contracts/TRUSTxGAMING.sol#657) shadows:
        - Ownable.owner() (contracts/TRUSTxGAMING.sol#306-308) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Pausable.Paused() (contracts/TRUSTxGAMING.sol#380-384) compares to a boolean constant:
        -require(bool)(canPause == true) (contracts/TRUSTxGAMING.sol#381)
Pausable.unpaused() (contracts/TRUSTxGAMING.sol#389-393) compares to a boolean constant:
        -require(bool)(pause == true) (contracts/TRUSTxGAMING.sol#390)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Context._msgData() (contracts/TRUSTxGAMING.sol#272-275) is never used and should be removed
SafeMath.div(uint256,uint256) (contracts/TRUSTxGAMING.sol#196-198) is never used and should be removed
SafeMath.div(uint256,uint256,string) (contracts/TRUSTxGAMING.sol#211-218) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/TRUSTxGAMING.sol#231-233) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/TRUSTxGAMING.sol#246-249) is never used and should be removed
SafeMath.mul(uint256,uint256) (contracts/TRUSTxGAMING.sol#171-183) is never used and should be removed
TRUSTxGAMING._mint(address,uint256) (contracts/TRUSTxGAMING.sol#614-621) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.5.17 (contracts/TRUSTxGAMING.sol#9) allows old versions
solc-0.5.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable Ownable._owner (contracts/TRUSTxGAMING.sol#290) is not in mixedCase
Function Pausable.Paused() (contracts/TRUSTxGAMING.sol#380-384) is not in mixedCase
Parameter TRUSTxGAMING.withdraw(address,address,uint256).con_address (contracts/TRUSTxGAMING.sol#667) is not in mixedCase
Parameter TRUSTxGAMING.airdrop(address[],uint256)._address (contracts/TRUSTxGAMING.sol#672) is not in mixedCase
Parameter TRUSTxGAMING.airdrop(address[],uint256)._amount (contracts/TRUSTxGAMING.sol#672) is not in mixedCase
Parameter TRUSTxGAMING.multiSender(address[],uint256[])._recipients (contracts/TRUSTxGAMING.sol#685) is not in mixedCase
Parameter TRUSTxGAMING.multiSender(address[],uint256[])._values (contracts/TRUSTxGAMING.sol#685) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (contracts/TRUSTxGAMING.sol#273)" inContext (contracts/TRUSTxGAMING.sol#263-276)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
TRUSTxGAMING.constructor() (contracts/TRUSTxGAMING.sol#423-430) uses literals with too many digits:
        - _totalSupply = 660000000000000000000000 (contracts/TRUSTxGAMING.sol#427)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
airdrop(address[],uint256) should be declared external:
        - TRUSTxGAMING.airdrop(address[],uint256) (contracts/TRUSTxGAMING.sol#672-680)
Moreover, the following function parameters should change its data location:
_address location should be calldata
multiSender(address[],uint256[]) should be declared external:
        - TRUSTxGAMING.multiSender(address[],uint256[]) (contracts/TRUSTxGAMING.sol#685-692)
Moreover, the following function parameters should change its data location:
_recipients location should be calldata
_values location should be calldata
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (6 contracts with 85 detectors), 27 result(s) found
```

Summary

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

Some issues of high, low and informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture. In The End, TRUSTxGaming Team has Acknowledged All Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the TRUSTxGAMING 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 TRUSTxGAMING 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.

850+
Audits Completed

$16B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2023

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com