QuillAudits

Audit Report
May, 2022

For

RONOSPAD

Table of Content

Executive Summary

Project Name	CronosPad
Overview	Cronospad is the first major micro and small cap decentralized launchpad on Cronos.
Timeline	May 10th, 2022 to May 16th, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse CronosPad codebase for quality, security, and correctness.
Sourcecode	*https://github.com/cronospad/cronospad*
Commit	1605d38fab314cab399093f756b41093fdbc859f
Fixed in	*https://github.com/cronospad/cronospad*
Commit	4f44581df5aa090764ec905121ee45889e796bdb

3
Issues Found

🟥 High 🟨 Medium
🟩 Low 🟪 Informational

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

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
- ✓ DoS with Block Gas Limit
- ✓ Transaction-Ordering Dependence
- ✓ Use of tx.origin
- ✓ Exception disorder
- ✓ Gasless send
- ✓ Balance equality
- ✓ Byte array
- ✓ Transfer forwards all gas
- ✓ ERC20 API violation
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Redundant fallback function
- ✓ Send instead of transfer
- ✓ Style guide violation
- ✓ Unchecked external call
- ✓ Unchecked math
- ✓ Unsafe type inference
- ✓ Implicit visibility leve

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 Analysis

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

Low Severity Issues

No issues were found

Informational Issues

A.1 Unlocked pragma (pragma solidity ^0.8.0)

Description

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 Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.eg 0.8.4

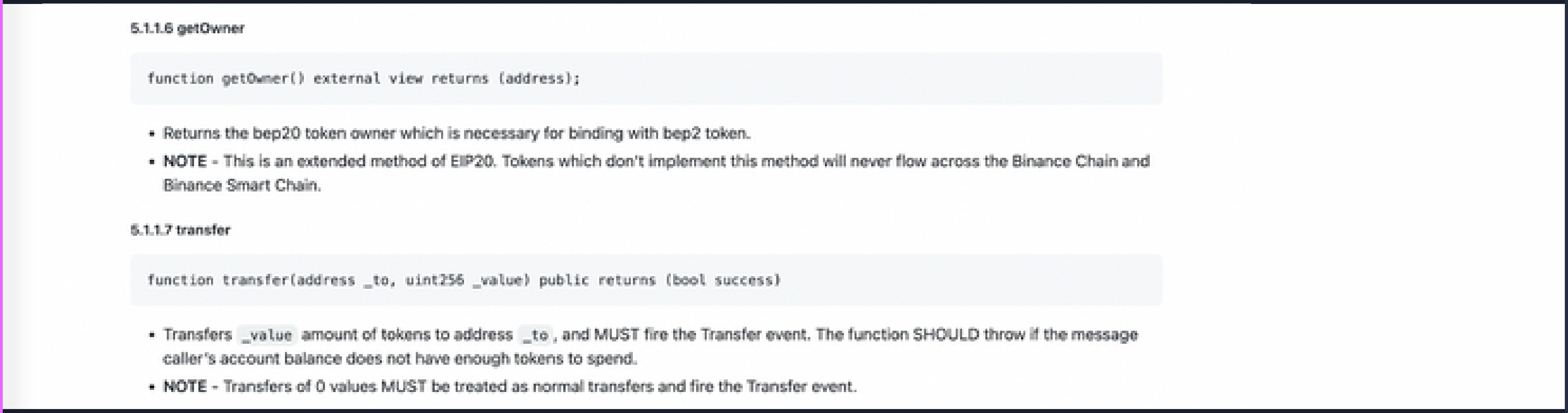
Status

Fixed

2. BEP20 Standard violation

Description

Implementation of transfer() function does not allow the input of zero amount as it's demanded in ERC20 and BEP20 standards. This issue may break the interaction with smart contracts that rely on full BEP20 support. Moreover, the GetOwner() function which is a mandatory function is missing from the contract.



5.1.1.6 getOwner

```
function getOwner() external view returns (address);
```

- Returns the bep20 token owner which is necessary for binding with bep2 token.
- **NOTE** - This is an extended method of EIP20. Tokens which don't implement this method will never flow across the Binance Chain and Binance Smart Chain.

5.1.1.7 transfer

```
function transfer(address _to, uint256 _value) public returns (bool success)
```

- Transfers `_value` amount of tokens to address `_to`, and MUST fire the Transfer event. The function SHOULD throw if the message caller's account balance does not have enough tokens to spend.
- **NOTE** - Transfers of 0 values MUST be treated as normal transfers and fire the Transfer event.

Recommendation

The transfer function must treat zero amount transfer as a normal transfer and an event must be emitted. Moreover, it is recommended to implement getOwner() function.
Reference: *https://github.com/bnb-chain/BEPs/blob/master/BEP20.md#5117-transfer*

Status

Fixed

3. Public functions that could be declared external inorder to save gas

Whenever a function is not called internally, it is recommended to define them as external instead of public in order to save gas. For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If your function is only called externally, then you should explicitly mark it as external. External function's parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge.
Here is a list of function that could be declared external:

- totalSupply()
- name()
- symbol()
- decimals()
- increaseAllowance()
- decreaseAllowance()

Status

Fixed

Functional Testing

Some of the tests performed are mentioned below

- ✓ Should be able call all getters
- ✓ Should be able to transfer token
- ✓ Should be able to approve
- ✓ Should be able to increaseApprove
- ✓ Should be able to decreaseApprove
- ✓ Should be able to transferFrom
- ✓ Should be able to burn token
- ✓ Should be able to burnFrom
- ✓ Should be able to transferOwnership
- ✓ Should revert if transfer amount exceeds balance

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

No Major Issues in Code, Just Some informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In End, CronosPad Team Resolved all Issues.

Disclaimer

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

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2022

For

RONOSPAD

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com