QuillAudits

Audit Report
April, 2022

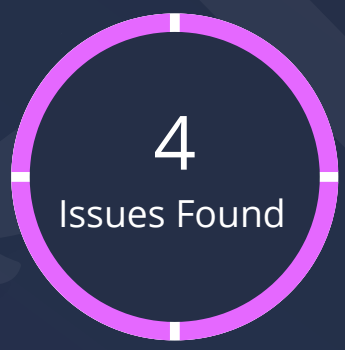
For

akt.io

Table of Content

Executive Summary

Project Name	Akt.io Token Contract
Overview	The Aktio Coin is a decentralised, peer-to-peer crypto asset that has been developed by Automata ICO Ltd. As an ERC-20 standard token. Built on the market-leading, Ethereum blockchain, AKTIO functions as a digital currency for multi-asset exchange and settlement of multi-sector, international transactions.
Timeline	April 7, 2022 - April 11, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Akt.io smart contract's codebase for quality, security, and correctness.
Source Code	*https://ropsten.etherscan.io/address/0x70cbf5e72a85b41808ef20f6eb05a9b8f223f220*
Fixed In	*https://ropsten.etherscan.io/token/0x02e64ded0c7acb0eb0fd495d957fb4c0bc25a24b*



High Medium

Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	1
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 Testing

A. Contract - CommonPausableERC20

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 Floating 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. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version get selected while deploying contract which has higher chances of having bugs in it.

Remediation

Lock the pragma version for the compiler version that is chosen.

Status

Fixed

A.2 Use of require statement instead of whenNotPaused modifier

```
928   function _beforeTokenTransfer(address from, address to, uint256 amount) internal override(ERC20) {
929     super._beforeTokenTransfer(from, to, amount);
930
931     require(!paused(), "ERC20Pausable: token transfer while paused");
932   }
```

Description

[#L928] _beforeTokenTransfer() uses a [#L931] require statement which works similar to whenNotPaused from a Pausable contract. Here whenNotPaused modifier can be used instead of using require check statement.

Remediation

Consider using whenNotPaused modifier to check and restrict _beforeTokenTransfer() from executing if the contract is paused.

Status

Fixed

A.3 approve(), increaseAllowance(), decreaseAllowance() works even if contract is paused

Description

In this case token holders can give/set token allowance to any address even when the contract is paused.

Remediation

Consider reviewing logic and adding restrictions to approve() in the case the contract is paused.

Status

Acknowledged

A.4 Incorrect error message while burning tokens if contract is paused

```
928  function _beforeTokenTransfer(address from, address to, uint256 amount) internal override(ERC20) {
929    super._beforeTokenTransfer(from, to, amount);
930
931    require(!paused(), "ERC20Pausable: token transfer while paused");
932  }
```

Description

If the contract is in paused state and someone calls burn function then the transaction will revert as expected but the error message that contract gives in this case can create confusion.

In the case of paused contract to restrict someone from calling some functions like transfer,transferFrom,mint,burn,The contract uses [#L931] require statement added in [#L928] _beforeTokenTransfer.

If condition fails it reverts with "ERC20Pausable: token transfer while paused".

While burning tokens [#L427] _burn calls overridden [#L928] _beforeTokenTransfer which will revert with "ERC20Pausable: token transfer while paused" error if contract is paused which might not be correct error message in this case and hence can create confusion.

Remediation

Consider reviewing logic about error messages.

Status

Fixed

Functional Testing

Some of the tests performed are mentioned below

- Should mint initial balance to msg.sender on deployment
- Should be able to mint tokens only by contract owner
- Should update balances of sender and recipient when token transferred.
- Should be able to approve tokens
- Should be able to increase allowance
- Should be able to decrease allowance
- Should be able to spend approved tokens
- Should be able to burn approved tokens
- Reverts while minting if totalsupply + amount to mint exceeds the cap limit
- Reverts while minting if _mintingFinished is true
- Reverts on transfer to zero address
- Reverts on approve to zero address
- Reverts if sender doesn't holds enough token balance for sending
- Reverts if spender doesn't holds enough approval to spend someone's tokens

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 Akt.io Smart Contract. We performed our audit according to the procedure described above.

The audit showed informational severity issues,at the end Majority of the issues are fixed by the Auditee.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Akt.io 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 Akt.io 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
April, 2022

For

akt.io

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com