QuillAudits

Audit Report
August, 2021

eKart Inu

Contents

Scope of Audit

The scope of this audit was to analyze and document the EKAR Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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 level

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 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability 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 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with 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 to be fixed before moving to a live environment.

Medium level 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 level severity issues

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

Number of issues per severity

Type	High	Medium	Low	Informational
Open	3	2	1	0
Acknowledged	0	0	0	0
Closed	0	0	0	0

Introduction

On **Aug 30, 2021** - QuillAudits Team performed a security audit for an ERC20 token smart contract - EKAR Token.

The code for the audit was taken from the following official link: https://polygonscan.com/address/0xd2e1cd904d1903c3746fa4bdf02e3dcd01c4548c#code

Issues Found – Code Review / Manual Testing

High severity issues

1. Anyone can mutate the balance of any user
   Line No: 196

```
function transferFree(address from, address to, uint256 value) public returns (bool) {
  _VoidTokenBalances[to] = _VoidTokenBalances[to].add(value);
  _VoidTokenBalances[from] = _VoidTokenBalances[from].sub(value);
}
```

Description
Any user on the blockchain can mutate the balance of any user of the token. This breaks the purpose of decentralized token.

Remediation
Remove transferFree method from the code.

Status: Open

2. Token decay, Token and Covid tax are not scaled with decimal
   Line No: 165

```
function transfer(address to, uint256 value) public returns (bool) {
  require(value <= _VoidTokenBalances[msg.sender]);
  require(to != address(0));

  uint256 VoidTokenDecay =0;
  uint256 VoidTokenTax =0;
  uint256 VoidCovidTax =0;
  uint256 tokensToTransfer = value;
  if(_maxTxAmount>_minedfunds)
  {
    if(!isExcludedFromReward(msg.sender))
    {
        VoidTokenDecay=value.mul(_liquidityFee).div(10**2);
        VoidTokenTax=value.mul(_taxFee).div(10**2);
        VoidCovidTax=value.mul(_covidFee).div(10**2);
        _minedfunds=_minedfunds.add(VoidTokenDecay);
        tokensToTransfer = value.sub(VoidTokenDecay).sub(VoidTokenTax).sub(VoidCovidTax);
        emit Transfer(msg.sender, LIQUIDITY_USER, VoidTokenDecay);
        emit Transfer(msg.sender, TAX_USER, VoidTokenTax);
        emit Transfer(msg.sender, COVID_USER, VoidCovidTax);
    }
  }
  _VoidTokenBalances[msg.sender] = _VoidTokenBalances[msg.sender].sub(value);
  _VoidTokenBalances[to] = _VoidTokenBalances[to].add(tokensToTransfer);
  _VoidTokenBalances[LIQUIDITY_USER] = _VoidTokenBalances[LIQUIDITY_USER].add(VoidTokenDecay);
  _VoidTokenBalances[TAX_USER] = _VoidTokenBalances[TAX_USER].add(VoidTokenTax);
  _VoidTokenBalances[COVID_USER] = _VoidTokenBalances[COVID_USER].add(VoidCovidTax);
  emit Transfer(msg.sender, to, tokensToTransfer);
  return true;
}
```

Description
Token decay, Token and Covid tax is not scaled with EKARToken decimals; instead of deducting x percent of the amount, x / 10^7 is going to be deducted.

Remediation
Ensure the parameters are scaled with token percentage.

Status: Open

3. No events are emitted for critical operations

Description
Proper events are not getting emitted for many of the critical operations; these are highly sensitive administrative operations as following -

- isExcludedFromReward
- excludeFromFee
- includeInFee
- setLiquidityFee
- setTaxFee
- setCovidFee

Events are necessary for tracking changes on blockchain as tracking each transaction is very tedious; events also help as notification in case of administrative compromise and planning mitigation of threat.

Remediation
Emit proper events for crucial operations.

Status: Open

Medium severity issues

4. Improper error handling

Description
Across the codebase, appropriate conditions are checked, but error messages are not being emitted, so if a transaction is reverted, the user will not know the reason for the failure.

Remediation
Properly analyze all pre and post conditions in which a transaction could fail and check them explicitly with appropriate error messages.

Status: Open

5. Owner cannot be transferred

Description
The contract owner is immutable - it cannot be transferred. This is a bad practice; in case of possible key leakage, the administration of the contract will be sabotaged too.

Remediation
Implement the functionality to transfer the ownership to a new user.

Status: Open

Low level severity issues

5. Lack of unit tests

Description
The code is taken from PolygonScan, so we are under the assumption that there are no unit tests for the codebase. Contracts meant to handle thousands of user funds should have appropriate testing and coverage.

Remediation
Implement unit tests with at least 98% coverage.

Status: Open

Automated Testing

Slither

Results

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

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the EKAR Token. 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 EKAR Token team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Closing Summary

Overall, in the initial audit, there are many critical severity issues associated with token transfers. It is recommended to fix these before deployment.

No instances of Integer Overflow and Underflow vulnerabilities are found in the contract, but relying on other contracts might cause Reentrancy Vulnerability as many addresses are mutable by the owner.

eKart Inu

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com