VIDMA

SMART CONTRACT AUDIT

Project: **Ammbr**

Date: **February 7th, 2022**

TABLE OF CONTENTS

SUMMARY

Vidma team has conducted a smart contract audit for the given codebase.

The contracts are in good condition. Based on the fixes provided by the Ammbr team and on the quality and security of the codebase provided, Vidma team can give a score of 95 to the audited smart contracts.

During the auditing process, the Vidma team has found a couple of informational issues, 7 issues with a low level of severity, 1 issue with a medium level of severity, and 3 issues with a critical level of severity.

Severity of the issue	Total found	Resolved	Unresolved
Critical	3 issues	3 issues	0 issues
High	0 issues	0 issues	0 issues
Medium	1 issue	1 issue	0 issues
Low	7 issues	7 issues	0 issues
Informational	5 issues	5 issues	0 issues
Total	**16 issues**	**16 issues**	**0 issues**

Evaluating the findings, we can assure that the contract is safe to use and all the issues found are performed only by certain conditions and cases. Under the given circumstances we can set the following risk level:

Vidma auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. Hence, it helps to adequately evaluate the development quality.

Based on the given findings, risk level, performance, and code style, Vidma team can grant the following overall score:

Vidma auditing team has conducted a bunch of integrated autotests to ensure that the given codebase has decent performance and security levels. The test results and the coverage can be found in the accompanying section of this audit report.

Please mind that this audit does not certify the definite reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by Vidma auditing team. If the code is under development, we recommend run one more audit once the code is finalized.

SCOPE OF WORK

Ammbr

They apply systems thinking to dynamically map out the influencing factors for each project. They establish relationships with partners who create added value and bring together the public and private sectors. They create cross-industry links for knowledge flows. They apply values that reinforce each other: socio-cultural value, experimental value, technical value, and economical value.

Within the scope of this audit, two independent auditors deeply investigated the given codebase and analyzed the overall security and performance of smart contracts.

The debrief took place from Jan 10th to 26th, 2022 and the final results are present in this document.

Vidma auditing team has made a review of the following contracts:

- Migrations;
- QC.

The source code was taken from the following **source**:
https://github.com/AmmbrFi/qc-contract/commit/4a3a40c6cd8a8756d4d214f30d85c1c6edb1a3cd

Initial commit submitted for the audit:
https://github.com/AmmbrFi/qc-contract/commit/4a3a40c6cd8a8756d4d214f30d85c1c6edb1a3cd

Last commit:
https://github.com/AmmbrFi/qc-contract/commit/03a39cfb80030ef5c9952d4ef54f228aaf86ecab

In order to conduct a more detailed audit, Ammbr has provided the following **documentation**:
https://github.com/AmmbrFi/qc-contract/blob/master/docs.md

WORKFLOW OF THE AUDITING PROCESS

During the manual phase of the audit, Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract.

Within the testing part, Vidma auditors run integration tests using the Truffle testing framework. The test coverage and the tests themselves are inserted into this audit report.

Vidma team uses the most sophisticated and contemporary methods and techniques to ensure the contract does not have any vulnerabilities or security risks:

- Re-entrancy;
- Access Management Hierarchy;
- Arithmetic Over/Under Flows;
- Unexpected Ether;
- Delegatecall;
- Default Public Visibility;
- Hidden Malicious Code;
- Entropy Illusion (Lack of Randomness);
- External Contract Referencing;
- Short Address/Parameter Attack;
- Unchecked CALL Return Values;
- Race Conditions / Front Running;
- General Denial Of Service (DOS);
- Uninitialized Storage Pointers;
- Floating Points and Precision;
- Tx.Origin Authentication;
- Signatures Replay;
- Pool Asset Security (backdoors in the underlying ERC-20).

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For the convenience of reviewing the findings in this report, Vidma auditors classified them in accordance with the severity of the issues. (from most critical to least critical). The acceptance criteria are described below.

All issues are marked as “Resolved” or “Unresolved”, depending on whether they have been fixed by Ammbr or not. The latest commit, indicated in this audit report should include all the fixes made.

To ease the explanation, the Vidma team has provided a detailed description of the issues and recommendations on how to fix them.

Hence, according to the statements above, we classified all the findings in the following way:

Finding	Description
Critical	The issue bear a definite risk to the contract, so it may affect the ability to compile or operate.
High	Major security or operational risk found, that may harm the end-user or the overall performance of the contract.
Medium	The issue affects the contract to operate in a way that doesn’t significantly hinder its performance.
Low	The found issue has a slight impact on the performance of the contract or its security.
Informational	The issue does not affect the performance or security of the contract/recommendations on the improvements.

MANUAL REPORT

Incorrect logic in *mint()* and *burn()* functions

Critical | Resolved

In *mint()* and *burn()* functions *_totalSupply* is changing but the value of *_gonsPerFragment* is not changing. It affects incorrect _gonBalances[user] calculation.

It triggers the following critical issues:

- When the user already has a significant amount of tokens on the balance, it is impossible to mint a certain amount to him. It happens because the new value of the user's balance exceeds the allocated limit for uint256 type variables.
- If mint or burn is called properly all balance calculations in all functions related to transfer are incorrect or even fail when the amount is too big because *_gonsPerFragment* is not valid according to the *_totalSuply*.

Steps to reproduce:

User balance: 100000000000000000000000 tokens.

Scaled balance: 1157920892373161954235709850086879078532699846656405640000000000000000000000

Minter try to mint 1 token for the user

Expected result:

- User balance: 100000100000000000000000 tokens
- Scaled balance: 1157920892373161954235709850086879078532699846656405637254480000000000000000

The real result:

Transaction failed because *gonValue* + *gonBalances[to]* in result this value is too large and can't be stored in the variable due to lack of dimensionality, which leads to transaction failure.

Recommendation:

Change the value of *_gonsPerFragment* variable to appropriate one every time you change the *_totalSuply* in *burn()* and *mint()* functions. As a suggestion possible solution to implement *mint()* function:

```
function mint(address to, uint256 amount) public
onlyRole(MINTER_ROLE) {
    require(to != address(0), "ERC20: mint to the zero address");
        uint256 balance = _gonBalances[to] / _gonsPerFragment;
        uint256 newBalance = balance + amount;
       _totalSupply += amount;
       _gonsPerFragment = TOTAL_GONS /_totalSupply;
       _gonBalances[to] = newBalance * _gonsPerFragment;
    }
```

Re-Audit:

Above solution was implemented and fixed the problem with overflow, but it triggers another problem which caused the incorrect rebase mechanizm. It is related to both *mint()* and *burn()* functions.

For example,

- user1 has 1m tokens in fragments on his balance, *totalSupply* - 1m tokens
- then mint 1m tokens for user2: user2 has 1m tokens on his balance, user1 has 2m tokens on his balance, totalSupply - 2m tokens. But expected result is user1 - 1m, user2 - 1m, totalSupply - 2m.

Re-Audit 2:

Removed mint and burn functionality.

Incorrect tax fee calculation in all functions related to token transfers

Critical | Resolved

In QC.sol contract when tokens are transferred there is some tax in tokens which is transferred to the *taxAddress* account. In some specific cases when the transfer amount is too big an unexpected revert appears and transactions fail. It happens in the line when the *taxFee* is calculated because the result of multiplication *ganValue* and tax variables exceeds the allocated limit for uint256 type variables.

Recommendation:

Consider fixing *taxFee* calculation in functions *transfer()*, *transferFrom()*, *transferAll()*. Possible solution (example for *transferAll()* function):

```
function transferAll(address to) external validRecipient(to) returns
(bool) {
        uint256 gonValue = _gonBalances[msg.sender];
        require(gonValue > 0, "ERC20: No balance to transfer");

        uint256 value = gonValue / _gonsPerFragment;

        delete _gonBalances[msg.sender];
        uint256 taxFee = gonValue / _gonsPerFragment * tax / 10000 *
        _gonsPerFragment;
        gonValue = gonValue - taxFee;

        _gonBalances[to] += gonValue;
        _gonBalances[taxAddress] += taxFee;

        emit Transfer(msg.sender, to, value);
        emit Tax(to, taxFee);
        return true;
    }
```

Contracts isn't compiled

Critical | Resolved

In contract QC in function *setTax()* use an undeclared variable *_tax* and as result the contract isn't compiled.

Recommendation:

Change name variable *_tax* for *_tax_ that* is declared as parameter in function.

Validation in the constructor and corresponding setter of *taxAddress* and tax values

Medium | Resolved

In the constructor of QC.sol contract the address for taxes transfer is set but there is no check if this address isn't a zero address.

In the constructor of QC.sol contract tax percentage amount is set. There is a possibility to set any amount even bigger than 100%.

Recommendation:

Consider adding additional checks/validation in the constructor:

- For *taxAddress* variable add check for zero address;
- Check if tax value isn't more than 100% + precision.

This recommendation is valid and requested for corresponding setters too.

Re-Audit:

It was fixed in the constructor only but not in the corresponding setters. Consider add same checking in the setters *setTax()* and *setAddress()*.

Re-Audit 2:

Fixed.

Unnecessary SafeMath library usage

Low | Resolved

At contract QC.sol you can avoid using openzeppelin's SafaMath, because with Solidity >= 0.8.0 arithmetic operations revert on underflow and overflow by default.

Recommendation:

Remove safe math import and replace .sub .add .div .mul with appropriate math operation to save gas usage for contract deployment and functions call with mathematical calls.

Functions visibility optimization

Low | Resolved

Functions *mint()*, *burn()* can be marked as external as they are not used in the contract.

Recommendation:

Consider changing visibility from public to external to safe gas usage on calling these functions.

Lack of check allowance while *transferFrom*

Low | Resolved

In functions *transferfrom*, *transferAllFrom* there is no check if the msg.sender has enough requested token allowance of 'from' user account.

Recommendation:

Consider adding additional requirements to check if there is enough allowance for msg.sender.

Lack of user balance check while transfer

Low | Resolved

In functions *transfer*, *transferAll* there is no check if the msg.sender account has enough balance for transfer of the requested tokens.

In functions *transferfrom*, *transferAllFrom* there is no check if the 'from' user account has enough balance for transfer of the requested tokens.

Recommendation:

Consider adding additional requirements to check if there is enough balance to transfer.

Unspecified visibility of variables

Low | Resolved

In lines 30, 31 there are variables without specified visibility:

```
address taxAddress;
uint256 tax;
```

Recommendation:

Explicitly specify the visibility.

Missed event emitting

Low | Resolved

In functions *mint* and *burn* tokens are transferred from and to some specifically requested address but there is no *Transfer()* event emitted.

Recommendation:

Consider adding the *Transfer()* event to the functions *mint* and *burn*.

Lack of zero-check in the function of QC.sol contract

Low | Resolved

In functions *approve*, *increaseAllowance*, *decreaseAllowance* there is no check if the ‘spender’ address isn’t zero address.

In functions *transferFrom*, *transferAllFrom* there is no check if the ‘from’ address parameter isn’t zero.

Recommendation:

Consider adding a check for zero address for functions *approve*, *increaseAllowance*, *decreaseAllowance*, *transferFrom* and *transferAllFrom*.

Lack of NatSpec annotations

Informational | Resolved

Smart contract QC.sol is not covered by NatSpec annotations.

Recommendation:

Consider to cover by NatSpec all contract`s methods.

Unused import

Informational | Resolved

In QC.sol contact in line 5 there is importing of ERC20Burnable.sol contract from openzeppelin but it is never used in the source code.

Recommendation:

Consider removing an unused import statement.

Requires without explanations

Informational | Resolved

Requires in the modifier *validRecipient()* of the QC contract without comments, and one condition is divided into two, which requires the use of more gas.

Recommendation:

Combine the conditions in the requirements and add the comment.

Re-Audit:

It was fixed by adding explanation messages but was left two require statements.

Functions order at QC.sol contract

Informational | Resolved

The functions in contract QC.sol are not grouped according to their visibility and order.

Functions should be grouped according to their visibility and ordered in the following way:

- Constructor;
- Receive function (if exists);
- Fallback function (if exists);
- External;
- Public;
- Internal;
- Private.

Recommendation:

Consider changing functions order according to solidity documentation: Order of Functions.

Order of Layout

Informational | Resolved

The layout contract elements in the QC contract are not logically grouped.

The contract elements should be grouped and ordered in the following way:

- Pragma statements;
- Import statements;
- Interfaces;
- Libraries;
- Contract.

Inside each contract, library or interface, use the following order:

- Library declarations (using statements);
- Constant variables;
- Type declarations;
- State variables;
- Events;
- Modifiers;
- Functions.

Ordering helps readers to navigate the code and find the elements more quickly.

Recommendation:

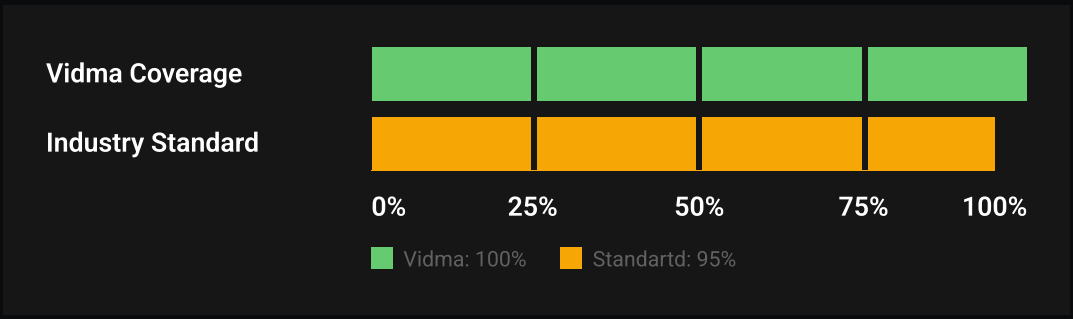
Consider changing the order of layout according to solidity documentation: Order of Layout.

TEST RESULTS

To verify the contract security and performance a bunch of integration tests was made using the Truffle testing framework.

Tests were based on the functionality of the code, business logic, and requirements and for the purpose of finding the vulnerabilities in the contacts.

In this section, we provide tests written by Vidma auditors.



It's important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the Ammbr repo. We write totally separate tests with code coverage of a minimum of 95%, to meet the industry standards.

Tests are written by Vidma

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
Migrations.sol	100.00	100.00	100.00	100.00
QC.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: QC
  QC Constructor Phase Test Cases
    ✓ should set name of the token correctly (159ms)
    ✓ should set symbol of the token correctly (128ms)
    ✓ should set total supply correctly (127ms)
    ✓ should set scaled balance of the owner correctly (126ms)
    ✓ should set balance of the owner correctly (140ms)
    ✓ should set roles for the owner correctly (544ms)
    ✓ should set address of the tax correctly (140ms)
    ✓ shouldn't set address of the tax if zero's address (627ms)
    ✓ should set value of the tax correctly (97ms)
    ✓ shouldn't set value of the tax if value less than 100 or
      more than 10000 (468ms)
  QC Set/Get Functions Phase Test Cases
    ✓ should get scalled total supply correctly (149ms)
    ✓ should set value of the tax correctly (522ms)
    ✓ shouldn't set value of the tax if caller haven't admin's
      role (392ms)
```

```
    ✓ shouldn't set value of the tax if value less than 100 or
      more than 10000 (249ms)
      should set address of the tax correctly (395ms)
  QC Rebase Function Phase Test Cases
      should rebase correctly if supplyDelta is zero (596ms)
      should rebase correctly if operation is zero (731ms)
      should rebase correctly if operation is one (735ms)
      should rebase correctly if _totalSupply more than
      MAX_SUPPLY (531ms)
      shouldn't rebase if caller haven't rebase's role (294ms)
      should get balances of users depending on rebase
      correctly (3060ms)
  QC Transfer Functions Phase Test Cases
      should transfer tokens correctly (1212ms)
      should transfer significant amount of tokens correctly (938ms)
      shouldn't transfer tokens if amount exceeds balance (547ms)
      shouldn't transfer tokens if recipient is zero's
      address (244ms)
    ✓ shouldn't transfer tokens if recipient is contract's
      address (283ms)
      should transfer all tokens correctly (1278ms)
    ✓ should transfer all tokens (significant amount)
      correctly (2206ms)
    ✓ shouldn't transfer all tokens if caller hasn't balance
      to transfer (763ms)
    ✓ should transfer tokens from other user correctly (1592ms)
    ✓ should transfer significant amount of tokens from other
      user correctly (1424ms)
    ✓ shouldn't transfer tokens from other user if amount
      exceeds allowance (606ms)
    ✓ shouldn't transfer tokens from other user if amount exceeds
      balance (833ms)
  QC Approve/Allowance Functions Phase Test Cases
    ✓ should approve tokens correctly (496ms)
    ✓ should increase allowance correctly (765ms)
    ✓ should decrease allowance correctly (755ms)
  Migrations Functions Phase Test Cases
    ✓ should set completed correctly (331ms)
    ✓ shouldn't set completed if caller isn't owner (310ms)

38 passing (28ms)
```

We are delighted to have a chance to work together with Ammbr team and contribute to their success by reviewing and certifying the security of the 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.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA