QUADENCY

SMART CONTRACT AUDIT

ZOKYO.

Sep 21st, 2021 | v. 1.0

PASS

Zokyo Security team has concluded that the given smart contracts passed security audit and are fully production-ready

SCORE

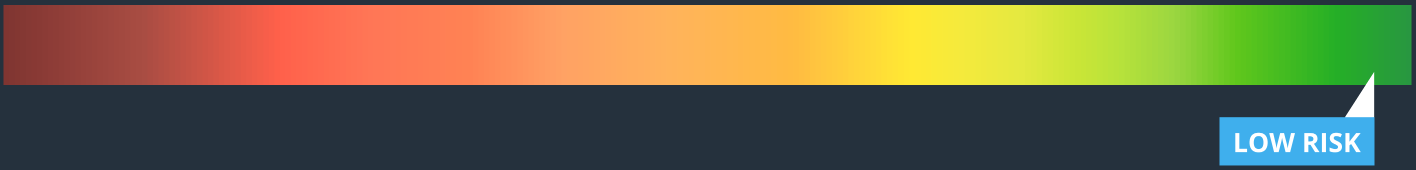
99

TECHNICAL SUMMARY

This document outlines the overall security of the Quadency smart contracts, evaluated by Zokyo's Blockchain Security team.

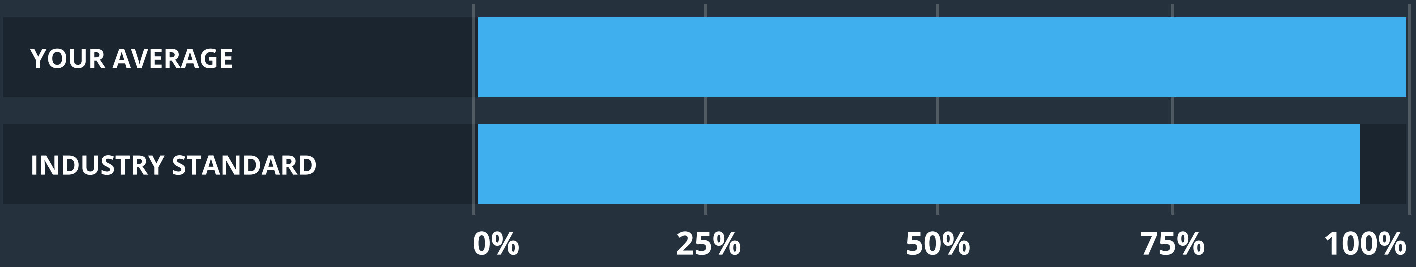
The scope of this audit was to analyze and document the Quadency smart contract codebase for quality, security, and correctness.

Contract Status



There were no critical issues found during the audit.

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Quadency team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Quadency repository.

Repository:
https://github.com/quadency/quad-token-contracts/commit/a86f4d1b051fcdff215b9e9c14717728b2224449

Last commit:
5c4b20ec01e0487307c4478b1820818a4cc6e9a9

Contracts under the scope:

- Migrations;
- MultiSigWallet;
- QUAD;
- TokenVesting;
- VestingFactory;
- WalletFactory.

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1. Due diligence in assessing the overall code quality of the codebase.
2. Cross-comparison with other, similar smart contracts by industry leaders.
3. Testing contract logic against common and uncommon attack vectors.
4. Thorough, manual review of the codebase, line-by-line.

SUMMARY

Zokyo team has conducted a smart contract audit for the given codebase. The scope of work and the contracts that are under the audit are presented in Auditing Strategy and Techniques Applied section.

During the auditing process, Zokyo auditing team has found informational issues and issues with the low severity level. No critical issues were spotted. It is worth mentioning that most of them are fixed by the Quadency team.

The contracts are in good condition. Based on the fixes provided by the Quadency team and on the quality and security of the codebase provided, Zokyo team can give a score of 99 to the audited smart contracts.

We can state that the contracts bear no security or operational risk to the end-user or contract owner, so they are fully production-ready.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged "Resolved" or "Unresolved" depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

● Critical

The issue affects the ability of the contract to compile or operate in a significant way.

● High

The issue affects the ability of the contract to compile or operate in a significant way.

● Medium

The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.

● Low

The issue has minimal impact on the contract's ability to operate.

● Informational

The issue has no impact on the contract's ability to operate.

COMPLETE ANALYSIS

Additional check is required for the constructor of QUAD.sol

LOW | RESOLVED

There is no verification for array length.

Recommendation:
Add check for arrays length:

```
require(
        wallets.length == amounts.length,
        "QUAD: wallets and amounts mismatch"
);
```

Additional check is required at TokenVesting.sol

LOW | RESOLVED

In function releaseVestedTokens() there is no verification for the zero address of the recipient.

Recommendation:
Add an additional check.

Order of Functions

INFORMATIONAL | UNRESOLVED

The functions in contract TokenVesting and MultiSigWallet 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.

Ordering helps readers identify which functions they can call and find the constructor and fallback definitions easier.

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

Order of Layout

INFORMATIONAL | RESOLVED

Layout contract elements in TokenVesting and MultiSigWallet contracts 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.
- token contract is not set to Bond & Unbond contracts

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.

Unnecessary check at TokenVesting.sol

INFORMATIONAL | RESOLVED

In function addRecipient (at line 114) and function addRecipientBatch (at line 170) of TokenVesting.sol you have check for parameter _startTime:

```
startTime: _startTime == 0 ? block.timestamp : _startTime,
```

But you already checked this parameter in lines 108, 109 (for function addRecipient) and in lines 157,158 (for function addRecipientBatch) according to them it should be greater than zero so it will always return false and will use _startTime.

Recommendation:
Remove unnecessary check:

```
startTime: _startTime,
```

Unnecessary require at MultiSigWallet.sol

INFORMATIONAL | RESOLVED

In the function tryInsertSequenceId() you require onlySigner. This function is private and can be called only with functions that already have this requirement.

Recommendation:
Remove the unnecessary requirements.

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

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Quadency team

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	87.26	60.71	86.11	86.14	
MultiSigWallet.sol	75.32	46.67	73.68	74.39	... 484, 485, 512
QUAD.sol	100.00	50.00	100.00	75.00	13
TokenVesting.sol	98.51	78.00	100.00	98.57	204
VestingFactory.sol	100.00	100.00	100.00	100.00	
WalletFactory.sol	100.00	100.00	100.00	100.00	
All files	**87.26**	**60.71**	**86.11**	**86.14**	

Test Results

```
Contract: MultiSigWalletUnitTest
  When sending token transactions
    ✓ should send single token transactions correctly (625ms)
    ✓ should send batch token transactions correctly (2203ms)
    ✓ should fail when other signer same as sender (1017ms)
    ✓ should fail when unauthorized sender but valid other signature (361ms)
    ✓ should fail when invalid signature but authorized sender (517ms)

Contract: TokenVestingUnitTest
  Initialization
    ✓ should set the Token correctly (96ms)
    ✓ should set the MultiSig correctly (86ms)
    ✓ should fail with 0 address for Token (500ms)
    ✓ should fail with 0 address for MultiSig (429ms)
    ✓ should fail when initialized second time (739ms)
  When setting vesting schedules
    ✓ should add new recipient correctly (647ms)
    ✓ should add new recipient batch correctly (1095ms)
    ✓ should fail when recipient batch mismatched (581ms)
    ✓ should fail when recipient already has vesting schedule (705ms)
    ✓ should fail when vesting duration zero (489ms)
    ✓ should fail when vesting cliff longer than vesting duration (490ms)
    ✓ should fail when vesting duration longer than 100 months (512ms)
    ✓ should fail when vested amount per month zero (500ms)
    ✓ should fail when start time is more than a year in past (536ms)
    ✓ should fail when start time is more than a year in future (502ms)
    ✓ should fail when called by address other than multisig (261ms)
  When claiming vested tokens
    ✓ should fail when releasing tokens before cliff reached (1300ms)
    ✓ should fail when releasing tokens before first month (1353ms)
    ✓ 6 months after vesting start date, user should be able to claim 6/24 of their total tokens (1832ms)
    ✓ 7 months after vesting start date, user should be able to claim 7/24 of their total tokens (1479ms)
    ✓ 8 months after vesting start date, user should be able to claim 8/24 of their total tokens (1337ms)
    ✓ 12 months after vesting start date, user should be able to claim 12/24 of their total tokens (1516ms)
    ✓ 18 months after vesting start date, user should be able to claim 18/24 of their total tokens (1822ms)
    ✓ 24 months after vesting start date, user should be able to claim 24/24 of their total tokens (1801ms)
    ✓ 1 months after vesting start date, user should be able to claim 1/6 of their total tokens (1763ms)
    ✓ 2 months after vesting start date, user should be able to claim 2/6 of their total tokens (1462ms)
    ✓ 3 months after vesting start date, user should be able to claim 3/6 of their total tokens (1840ms)
    ✓ 4 months after vesting start date, user should be able to claim 4/6 of their total tokens (1372ms)
    ✓ 5 months after vesting start date, user should be able to claim 5/6 of their total tokens (1827ms)
    ✓ 6 months after vesting start date, user should be able to claim 6/6 of their total tokens (1607ms)
    ✓ 1 months after vesting start date, user should be able to claim 1/16 of their total tokens (1558ms)
    ✓ 10 months after vesting start date, user should be able to claim 10/21 of their total tokens (1761ms)
    ✓ 21 months after vesting start date, user should be able to claim 21/26 of their total tokens (1640ms)
    ✓ 26 months after vesting start date, user should be able to claim 26/35 of their total tokens (1593ms)
    ✓ 29 months after vesting start date, user should be able to claim 29/38 of their total tokens (1730ms)
    ✓ 25 months after vesting start date, user should be able to claim 25/49 of their total tokens (1735ms)
    ✓ should successfully release batches of vested tokens (14960ms)

 42 passing (2m)
```

Tests written by Zokyo Security team

As part of our work assisting Quadency in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Quadency contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	100.00	100.00	100.00	100.00	
MultiSigWallet.sol	100.00	100.00	100.00	100.00	
QUAD.sol	100.00	100.00	100.00	100.00	
TokenVesting.sol	100.00	100.00	100.00	100.00	
VestingFactory.sol	100.00	100.00	100.00	100.00	
WalletFactory.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**100.00**	**100.00**	**100.00**	

Test Results

```
Contract: Migrations
    Migrations Functions Phase Test Cases
        ✓ should set completed correctly (518ms)
        ✓ shouldn't set completed if msg.sender isn't owner (797ms)

Contract: WalletFactory
    alletFactory Functions Phase Test Cases
        ✓ should create wallet correctly (657ms)
```

```
      ✓ shouldn't create wallet if count of signers less than 3 (159ms)
      ✓ shouldn't create wallet if one of the signers has zero's address (347ms)

  Contract: QUAD/VestingFactory
    VestingFactory Functions Phase Test Cases
      ✓ should create vesting contract correctly (823ms)
    QUAD Constructor Phase Test Cases
      ✓ should get token's name correctly (158ms)
      ✓ should get token's symbol correctly (209ms)
      ✓ shouldn't deploy if wallets and amounts mismatch (571ms)
      ✓ shouldn't mint on more than 10 wallets (1976ms)

  Contract: TokenVesting
    TokenVesting Functions Phase Test Cases
      ✓ should init correctly (712ms)
      ✓ shouldn't init if address of token or address of multiSigContract is zero (979ms)
      ✓ shouldn't init if contract already initialized (1185ms)
      ✓ should add recipient correctly (1099ms)
      ✓ shouldn't add recipient if msg.sender is unauthorized address (1184ms)
      ✓ shouldn't add recipient if recipient already added (1370ms)
      ✓ shouldn't add recipient if vesting duration is zero (436ms)
      ✓ shouldn't add recipient if vesting duration more than 100 (564ms)
      ✓ shouldn't add recipient if vesting cliff more than vesting duration (501ms)
      ✓ shouldn't add recipient if start time more than a year in future (511ms)
      ✓ shouldn't add recipient if start time more than a year in past (498ms)
      ✓ shouldn't add recipient if amount is zero (567ms)
      ✓ should get vesting schedule correctly (2100ms)
      ✓ should add recipient's batch correctly (1609ms)
      ✓ shouldn't add recipient's batch if recipients and amounts mismatch (504ms)
      ✓ shouldn't add recipient's batch if vesting duration is zero (527ms)
      ✓ shouldn't add recipient's batch if vesting duration more than 100 (347ms)
      ✓ shouldn't add recipient's batch if vesting duration less than vesting cliff (400ms)
      ✓ shouldn't add recipient's batch if start time more than a year in future (426ms)
      ✓ shouldn't add recipient's batch if start time more than a year in past (541ms)
      ✓ shouldn't add recipient's batch if one's of recipients address is zero (870ms)
      ✓ shouldn't add recipient's batch if recipient already added (1679ms)
      ✓ shouldn't add recipient's batch if amount is zero (949ms)
      ✓ should return as result '0, 0' if startTime has passed (1035ms)
      ✓ should return as result '0, 0' if cliff was reached (1367ms)
      ✓ should return as result 'monthsVested, amountVested' if over vesting duration (1199ms)
      ✓ should return as result 'vestingDuration, amount' if over vesting duration (1103ms)
      ✓ should transfers vested tokens correctly (2234ms)
      ✓ shouldn't transfer vested tokens if vested tokens are zero (2064ms)
      ✓ shouldn't transfer vested tokens if not enough recipients (640ms)
      ✓ shouldn't transfer vested tokens if recipients more than 255 (469ms)

  Contract: MultiSigWallet
    MultiSigWallet Functions Phase Test Cases
      ✓ should init correctly (831ms)
      ✓ shouldn't init if contract already initialized (678ms)
      ✓ should send with multi-signature correctly (769ms)
      ✓ shouldn't send with multi-signature if invalid signature 's' value (622ms)
      ✓ shouldn't send with multi-signature correctly if expire time less than current time (627ms)
      ✓ shouldn't send with multi-signature correctly if msg.sender is non-signer (380ms)
      ✓ shouldn't send with multi-signature correctly if we aren't in safe mode (1210ms)
      ✓ shouldn't send with multi-signature if signature haven't correct size (487ms)
      ✓ shouldn't send with multi-signature if transaction execution failed (882ms)
      ✓ shouldn't send with multi-signature if id more than maximum (2823ms)
      ✓ shouldn't send with multi-signature if id already used (648ms)
      ✓ shouldn't send with multi-signature if sequence id below window (8845ms)
      ✓ shouldn't send with multi-signature if signature isn't sign of signer (1370ms)
      ✓ shouldn't send with multi-signature if signers can't be equal (827ms)
      ✓ should send multiSig batch correctly (816ms)
      ✓ shouldn't send multiSig batch if not enough recipients (731ms)
      ✓ shouldn't send multiSig batch if size of arrays is unequal (523ms)
      ✓ shouldn't send multiSig batch if recipients more than 255 (2048ms)
      ✓ shouldn't send multiSig batch if we aren't in safe mode (1748ms)
      ✓ shouldn't send multiSig batch if insufficient funds (611ms)
      ✓ shouldn't send multiSig batch if call failed (1596ms)
      ✓ should send token with multiSig correctly (1089ms)
      ✓ should send token batch with multiSig correctly (1301ms)
      ✓ shouldn't send token batch with multiSig if not enough recipients (679ms)
      ✓ shouldn't send token batch with multiSig if size of arrays is unequal (563ms)
      ✓ shouldn't send token batch with multiSig if recipients more than 255 (817ms)
      ✓ shouldn't send token batch with multiSig if we aren't in safe mode (578ms)
      ✓ shouldn't send token batch with multiSig if insufficient funds (410ms)
```

```
        ✓ get call receive function correctly (1878ms)
        ✓ get call fallback function correctly (911ms)

  71 passing (8m)
```

We are grateful to have been given the opportunity to work with the Quadency team.

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.

Zokyo's Security Team recommends that the Quadency team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.