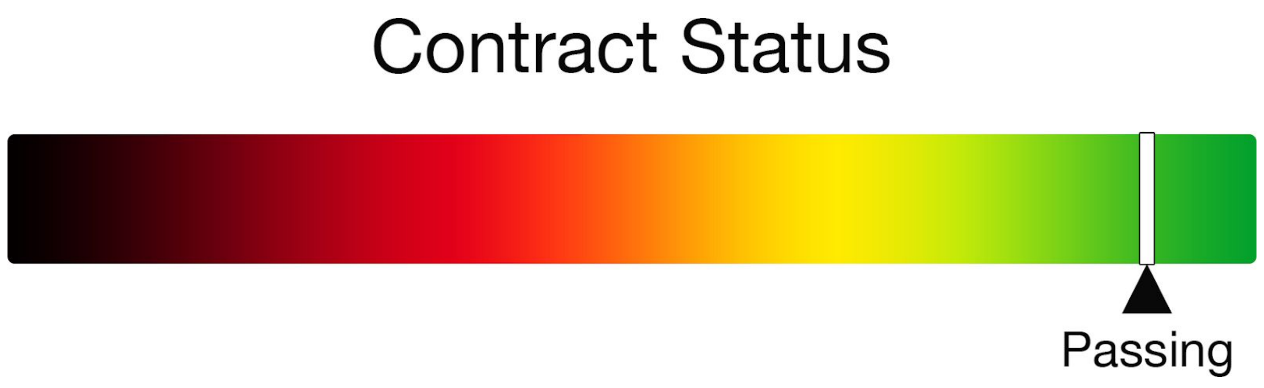
Quadrant Protocol Contract Audit

Prepared by Hosho
May 2018

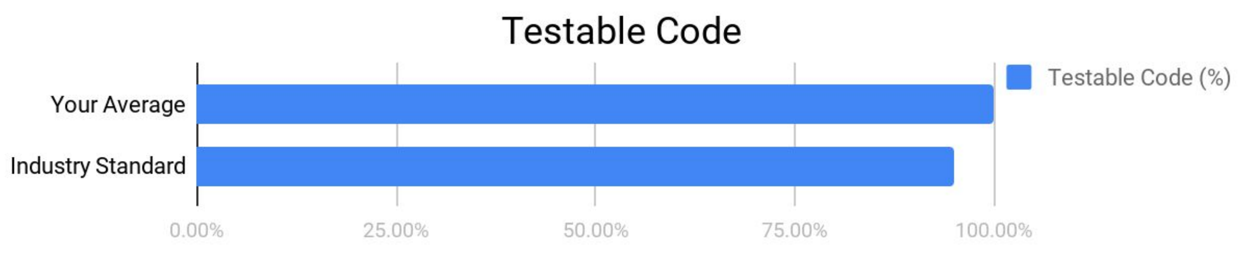
Report Version: 2.0

Executive Summary

This document outlines the overall security of Quadrant Protocol's smart contract as evaluated by Hosho's Smart Contract auditing team. The scope of this audit was to analyze and document Quadrant Protocol's token contract codebase for quality, security, and correctness.



All issues have been remediated. (See Complete Analysis)



Testable code is 100% which is higher than industry standard. (See Coverage Report)

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 Hosho recommend that the Quadrant Protocol Team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table Of Contents

1. Auditing Strategy and Techniques Applied

The Hosho Team has performed a thorough review of the smart contract code, the latest version as written and updated on April 30, 2018. All main contract files were reviewed using the following tools and processes. (See All Files Covered)

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

- Implements and adheres to existing ERC-20 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; and
- Is not affected by the latest vulnerabilities.

The Hosho Team has followed best practices and industry-standard techniques to verify the implementation of Quadrant Protocol's token contract. To do so, the code is reviewed line-by-line by our team of expert pentesters and 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.
5. Deploying the smart contract to testnet and production networks using multiple client implementations to run live tests.

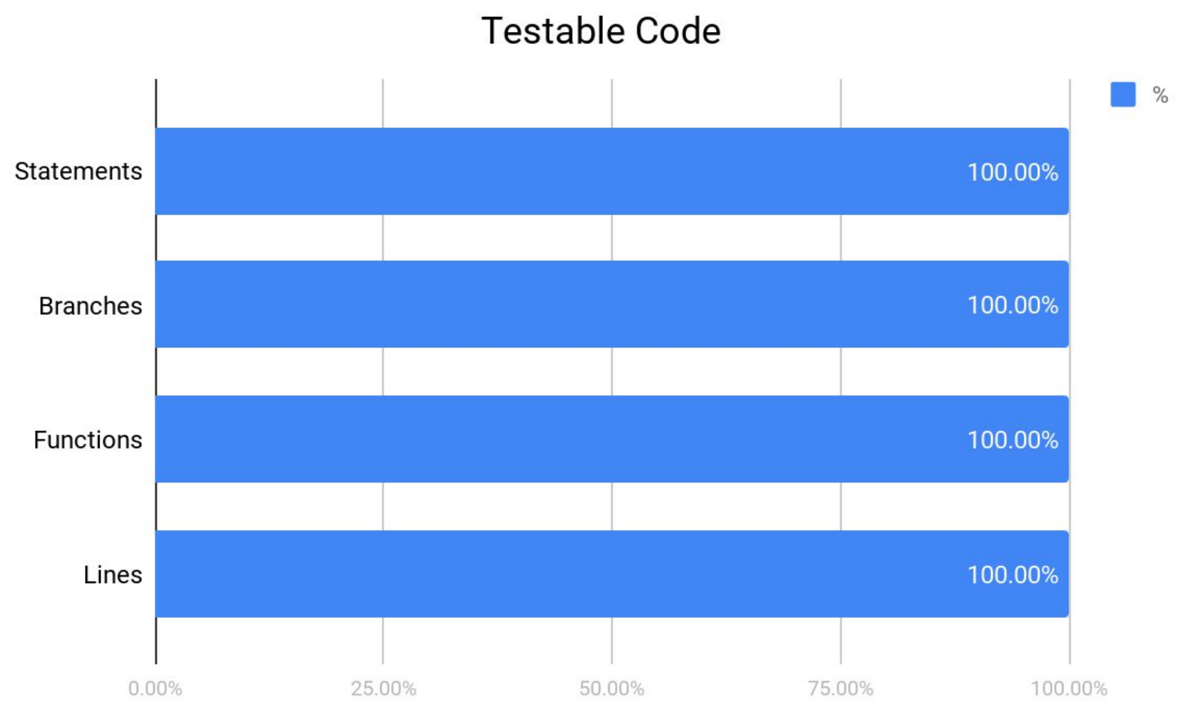
2. Structure Analysis and Test Results

2.1. Summary

The Quadrant Protocol contracts comprise a token named QuadToken and a token sale, QuadTokenSale. The token is ERC-20 compliant and the token sale is a flat-rate, standard sale contract. There is additional functionality added to allow administrators to alter the ETH to token ratio as needed.

2.2 Coverage Report

As part of our work assisting Quadrant Protocol in verifying the correctness of their contract code, our team was responsible for writing a unit test suite using the Truffle testing framework.



For each file see Individual File Coverage Report

2.3 Failing Tests

No failing tests.

See Test Suite Results for all tests.

3. Complete Analysis

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 still need addressing. 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 contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.
- **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, and is meant only as additional information.

1.1 Resolved, Low: Unnecessary Parameter

QuadTokenSale

Explanation

The `buy` function in this contract has `beneficiary` as a parameter but only checks that it is not a 0 address and is not used after. The rest of the function interacts with msg.sender exclusively, making this parameter and check unnecessary.

Resolution

The Quadrant Protocol Team has removed the `beneficiary` parameter, resolving this issue.

1.2 Resolved, Low: Missing Event

QuadTokenSale

Explanation

In the `withdrawToken` function, on line 101, there is a call to the event named `logWithdrawal`. Since the return statement is executed prior to this call, the event is never fired. If this event is required for logging purposes or for interactions with 3rd parties, this missed event will cause inaccurate reporting.

Resolution

The `logWithdrawal` event has been placed before the return statement by the Quadrant Protocol Team, ensuring that this event fires and resolving this issue.

1.3. Resolved, Low: Token Transfer Failure

QuadTokenSale

Explanation

The current design of the `buy` function will accept ETH even if the transfer function fails. To protect the user interacting with the contract, a revert should be considered to return the ETH in the case of failure.

Resolution

The Quadrant Protocol Team has protected the token transfer with a require statement that will revert in case of failure, resolving this issue.

1.4. Resolved, Informational: Modified Standard Contracts

BasicToken

Explanation

The OpenZeppelin contracts in this codebase, such as BasicToken, have been modified which makes it difficult to verify the correct operation of library contracts. It is recommended to add the modifiers to non-standard contracts, like QuadToken, and wrap calls to super.transfer as needed.

Resolution

Acknowledged by the Quadrant Protocol Team and not implemented due to necessary extensive code changes.

1.5. Resolved, Informational: No Token Escape

QuadTokenSale

Explanation

The Hosho Team suggests adding an escape function for trapped tokens that are not issued by the contract. There are an increasing number of ERC-20 and ERC-223 tokens, some of very large value, getting trapped forever in contracts, so it is valuable to have a function that can return these tokens to a contract issuer or owner for refund.

Resolution

Acknowledged by the Quadrant Protocol Team and not implemented due to necessary extensive code changes.

---

4. Closing Statement

We are grateful to have been given the opportunity to work with the Quadrant Protocol Team.

The team of experts at Hosho, having backgrounds in all aspects of blockchain, cryptography, and cybersecurity, can say with confidence that the Quadrant Protocol contract is free of any critical issues.

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.

We at Hosho recommend that the Quadrant Protocol Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties

5. Appendix A

Test Suite Results

Coverage Report

Contract: Crowdsale Tests for QuadToken

✓ Should require day zero limit to be greater than zero (42ms)

✓ Should deploy the crowdsale with the proper configuration (108ms)

✓ Should check crowdsale token withdraw for owner (255ms)

✓ Should log withdrawal event (202ms)

✓ Should verify changeLimit functions (255ms)

✓ Should verify enable functions (120ms)

✓ Should verify enable and whitelist functions (278ms)

✓ Should require sales to be enabled for buying tokens (171ms)

✓ Should test the buy function (722ms)

✓ Should not limit token purchases per day after day 2 (228ms)

✓ Should allow buying via fallback function (228ms)

✓ Should verify kill function (194ms)

✓ Should require buyer list to be smaller than 100000 when adding users (187ms)

✓ Should require buyer list to be smaller than 100000 when removing users (81ms)

Contract: ERC-20 Tests for QuadToken

✓ Should deploy a token with the proper configuration (40ms)

✓ Should allocate tokens per the minting function, and validate balances (316ms)

✓ Should transfer tokens from 0xd86543882b609b1791d39e77f0efc748dfff7dff to 0x42adbad92ed3e86db13e4f6380223f36df9980ef (49ms)

✓ Should not transfer negative token amounts

✓ Should not transfer more tokens than you have

✓ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens

✓ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization

```
  ✓ Should allow 0x667632a620d245b062c0c83c9749c9bfadf84e3b to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and 0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to 0x341106cb00828c87cd3ac0de55eda7255e04933f (215ms)

  ✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer negative tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b

  ✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b to 0x0

  ✓ Should not transfer tokens to 0x0

  ✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer more tokens than authorized from 0x667632a620d245b062c0c83c9749c9bfadf84e3b

  ✓ Should allow an approval to be set, then increased, and decreased (117ms)

Contract: SafeMath

  ✓ Should skip operation on multiply by zero

  ✓ Should revert on multiply overflow

  ✓ Should allow regular multiply

  ✓ Should revert on divide by zero

  ✓ Should allow regular division

  ✓ Should revert on subtraction overflow

  ✓ Should allow regular subtraction

  ✓ Should revert on addition overflow

  ✓ Should allow regular addition

Contract: Ownership Tests for QuadToken

  Deployment

    ✓ Should deploy with the owner being the deployer of the contract

  Transfer

    ✓ Should not allow a non-owner to transfer ownership

    ✓ Should not allow the owner to transfer to 0x0

    ✓ Should allow the owner to transfer ownership (40ms)
```

Vulnerabilities Tested

The vulnerabilities that have been tested for these contracts include, but are not limited to:

- Overflow attacks
- Underflow attacks
- Permissions exploits
- Reentrancy (DAO)
- Batch Overflow
- Unlimited Token Minting
- Insecure delegateCall()
- Insecure tx.origin
- Inheritance exploits
- Logic attacks

6. Appendix B

All Contract Files Tested

Commit Hash: c30b4792cd65c730202db96796eb391d225be90e

File	Fingerprint (SHA256)
contracts/BasicToken.sol	77937034b409a77cd522dcd0b45396c416cfbe864d6b1fb45bc91789f53419f4
contracts/ERC20.sol	6b75acd05c29968b057ec1facf659c064dbe0a79ac01444530629f01ef3a3abf
contracts/ERC20Basic.sol	86c0a5fc6cb564ae77140da57a8ff9a22f46404240e69a6782ff741e286d373a
contracts/Ownable.sol	35dcf237365077adb1dd8d1da9e05f2b4f8e9d7b49311fc8a09b28d4ce191579
contracts/QuadToken.sol	f6321d67346a1b596f3d837846b33c25b5007673d327f4d7ba628dde4f4b5a5e
contracts/QuadTokenSale.sol	248581c7ffebf9cd7a6d663ddc68a7df637633d8f21730e0a933eb2d9566679d
contracts/SafeMath.sol	e434336813af116101008bcfaed8cc02fa051c9c2b612a477b6bfa0765fa17f6
contracts/StandardToken.sol	ebd760bbf65146dad56a335bfdaa040537aa0f196d11dc4f34dcdb11d601ecea

7. Appendix C

Individual File Coverage Report

File	**% Statements**	**% Branches**	**% Functions**	**% Lines**
contracts/BasicToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/ERC20.sol	100.00%	100.00%	100.00%	100.00%
contracts/ERC20Basic.sol	100.00%	100.00%	100.00%	100.00%
contracts/Ownable.sol	100.00%	100.00%	100.00%	100.00%
contracts/QuadToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/QuadTokenSale.sol	100.00%	100.00%	100.00%	100.00%
contracts/SafeMath.sol	100.00%	100.00%	100.00%	100.00%
contracts/StandardToken.sol	100.00%	100.00%	100.00%	100.00%
All files	**100.00%**	**100.00%**	**100.00%**	**100.00%**