RepuX Token Contract Audit

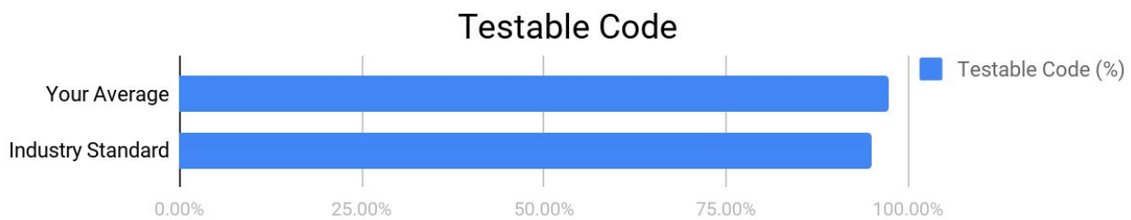
by Hosho, February 2018

NOTICE: *This document is a draft and is not representative of a completed audit.*

Executive Summary

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

All issues have been corrected and there are no failing tests. (See Complete Analysis.)



The percentage of testable code is above industry standard. (See Coverage Report.)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, it is merely an assessment of its 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 RepuX 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 as written and last updated on January 27, 2018. All main contract files were reviewed using the following tools and processes. (See Contract Files Tested.)

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

- Implements and adheres to existing ERC-20 Token standard 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.

The Hosho Team has followed best practices and industry-standard techniques to verify the implementation of RepuX's token contract. Our staff of expert pentesters and smart contract developers reviewed the contract line by line , documenting any issues as they were discovered. Part of this work included writing a code-specific unit test suite using the Truffle testing framework. As demonstrated, our strategies consist largely of manual collaboration between multiple team members at each stage of the review, including:

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. A 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.

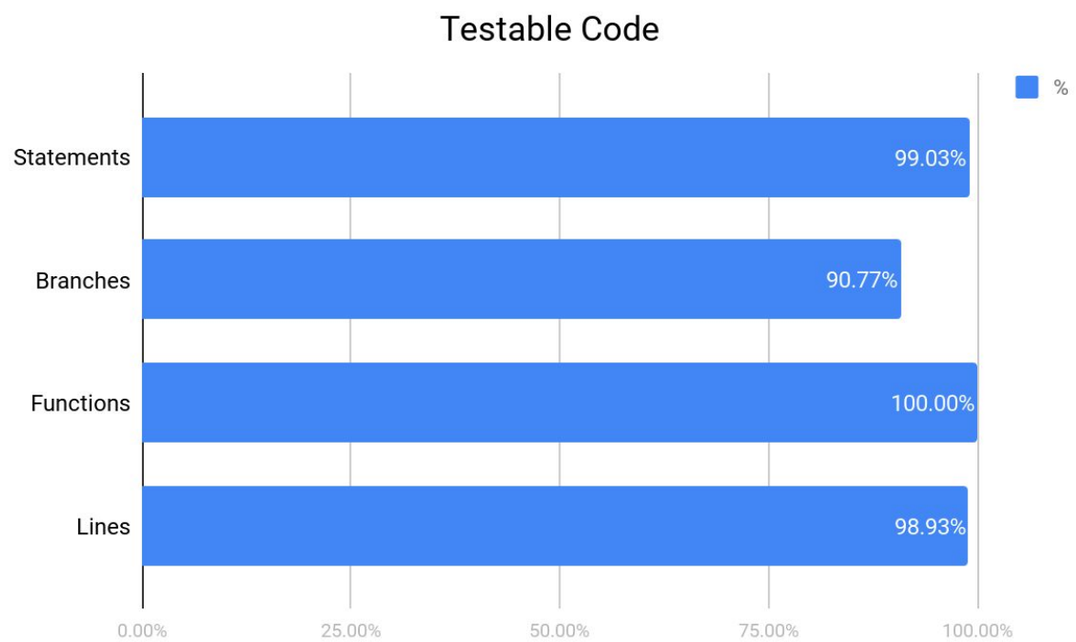
2. Structure Analysis and Test Results

2.1. Summary

The RepuX contract is a combined ERC-20/Crowdsale contract composed around the functionality of tracking blockID's. This allows alterations to timing in the event that there are any drastic changes within the ETH ecosystem in regards to block timing. The testable code remains very high and there are no failing tests.

2.2 Coverage Report

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



For individual files see Additional 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 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 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.

3.1. Resolved, Critical: Integer Overflow

Explanation

Using the `decimals` variable, a uint8, for exponential multiplication (10 ** `decimals`) causes a wrap around, or overflow, error. This sets the value to zero which then causes a cascade of multiple values being set to zero. As we implemented in our tests, if the desired value is to be 10^18, we would recommend multiplying by 1 ether to achieve this.

Resolution

The contract has been updated by the RepuX Team to multiply by 1 ether instead of (10 ** `decimals`), eliminating the integer overflow.

3.2. Resolved, Critical: Token Value Mis-issuance

Explanation

The issued tokens are not multiplied by the decimal value set for the tokens which results in 18 orders of magnitude less tokens than expected.

Resolution

The tokens are now being multiplied by the decimal value creating the proper number of tokens.

3.3. Resolved, Informational: Event Value Issuance

Explanation

On lines 495 and 512, the parameters were updated from `c.ethWei` to `msg.value`. As `msg.value` represents the value sent in by the command issuer, it does not equal the ETH amount that was paid into the contract from that transaction hash.

Resolution

The RepuX Team reverted this change back to `c.ethWei` from `msg.value` once again providing the proper value to this contract.

3.4. Resolved, Informational: ERC-20 Non-Compliance

Explanation

The declaration for `decimals` should be a uint8 as opposed to uint256.

Resolution

The `decimals` constant is now being initialized as a uint8 bringing the contract closer to compliance by declaring variables of the appropriate size as detailed in the ERC-20 Standards.

3.5. Resolved, Informational: ERC-20 Non-Compliance

Explanation

All issuance, including initial setup, should emit an event for tracking purposes.

Resolution

There are now events firing for all occurrences of issuance, bringing the overall contract closer to compliance with the ERC-20 Standards.

3.6. Resolved, Informational: Lack of Lockout

Explanation

There is no locking ability built into the `setBasePrice` function allowing the price to be altered at any point during the sale. While not a security issue, due to the `onlyOwner` modifier, this could cause potential investors concern.

Resolution

The RepuX Team added `require(!crowdsaleStarted);` to prevent the price from being adjusted once the crowdsale has begun.

3.7. Resolved, Informational: Team Lockout Not Utilized

Explanation

The team withdrawal functionality was updated to no longer utilize the teamLockUp variable. Despite the removal, the variable is still being declared and set within the contract. If there is no intended use for this variable it should be removed from the contracts.

Resolution

The `teamLockUp` has been added to the `withdrawTeamToken` by the RepuX Team.

---

4. Closing Statement

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

Overall the RepuX contract is a well developed contract with a high area of testable code. The RepuX Team has been responsive and eager to implement the corrections and suggestions made by the Hosho Team. As a small team of experts, having backgrounds in all aspects of blockchain, cryptography, and cybersecurity, we can say with confidence that the RepuX 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 RepuX Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

5. Test Suite Results

For the purposes of testing, the following changes were made to the RepuX contract:

- Base price set to 1 Finney
- `transferLockup` set to 10 blocks
- `averageBlockTime` set to 86400

Contract: ERC-20 Compliant Token

✓ Should deploy with RepuX as the name of the token

✓ Should deploy with REPUX as the symbol of the token

✓ Should deploy with 18 decimals

✓ Should deploy with 5e+26 tokens

✓ Should allocate tokens per the minting function, and validate balances (12095ms)

✓ Should transfer tokens from 0xd86543882b609b1791d39e77f0efc748dfff7dff to 0x42adbad92ed3e86db13e4f6380223f36df9980ef (84ms)

✓ Should not transfer negative token amounts

✓ Should not transfer more tokens than you have

✓ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens (38ms)

✓ Should not allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer an additional 1000 tokens once authorized, and authorization balance is > 0

✓ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization (43ms)

✓ Should allow 0x667632a620d245b062c0c83c9749c9bfadf84e3b to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and 0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to 0x341106cb00828c87cd3ac0de55eda7255e04933f (147ms)

✓ 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 (159ms)

Contract: RepuX Crowdsale

✓ Should not allow minting by the owner, until presale starts (59ms)

✓ Should not allow the presale to be started by anyone other than the owner

✓ Should allow the owner to halt (38ms)

✓ Should allow the owner to start the presale (41ms)

✓ Should block funds transfers while halted

✓ Should only allow the owner to start the presale once

✓ Should allow the owner to unhalt the contract (43ms)

✓ Should allow minting by the owner, under the proper settings - During presale (59ms)

✓ Should accept presale transfers when not halted - for acceptance (121ms)

✓ Should accept presale transfers when not halted - for rejection (118ms)

✓ Should let the owner conclude the presale (51ms)

✓ Should let the owner conclude the presale only once

✓ Should not permit buys with no value

✓ Should not permit buy outside of crowdsale/presale

✓ Should allow the acceptance of a contribution (74ms)

✓ Should only allow a contribution to be accepted once (46ms)

✓ Should not allow the rejection of an accepted contribution

✓ Should allow the owner to set a new multisig address (40ms)

✓ Should not allow the owner to set a new multisig address of 0x0

✓ Should not allow the owner to set a new average block time of 0

✓ Should allow the owner to set a new average block time (52ms)

✓ Should let the owner set a new owner, and be accepted by the new candidate (146ms)

✓ Should not let the owner attempt to set the owner to null

✓ Should not let the owner attempt to set the new owner to themselves

✓ Should not let anyone but the `ownerCandidate` accept the transfer

✓ Should not permit transfers before the 48 hour holding period at end of crowdsale is up

✓ Should permit the starting of the crowdsale (100ms)

✓ Should only permit the starting of the crowdsale once

✓ Should allow minting by the owner, under the proper settings (104ms)

✓ Should issue tokens through the phases (No value confirms happening here) (2513ms)

✓ Should validate the correct balances at each contribution stage (158ms)

✓ Should allow the rejection of a presale contribution (72ms)

✓ Should allow the rejection of a crowdsale contribution (63ms)

✓ Should only allow a contribution to be rejected once (46ms)

✓ Should not allow the acceptance of a rejected contribution

✓ Should allow the crowdsale to be ended (62ms)

✓ Should only allow the crowdsale to be ended once

✓ Should not allow minting by the owner, once the sale ends

✓ Should allow the owner to burn tokens (44ms)

✓ Should allow the base price to be adjusted (54ms)

✓ Should allow the team withdrawal to happen under the correct circumstances (293ms)

6. All Contract Files Tested

File	Fingerprint (SHA256)
Repux.sol	6e704acb514fc719e8f16556da9b7bcc43ca3ff0de2e47a1258006d46801461e

7. Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
contracts/Repux.sol	99.03%	90.77%	100.00	98.93%
All files	**99.03%**	**90.77%**	**100.00**	**98.93%**