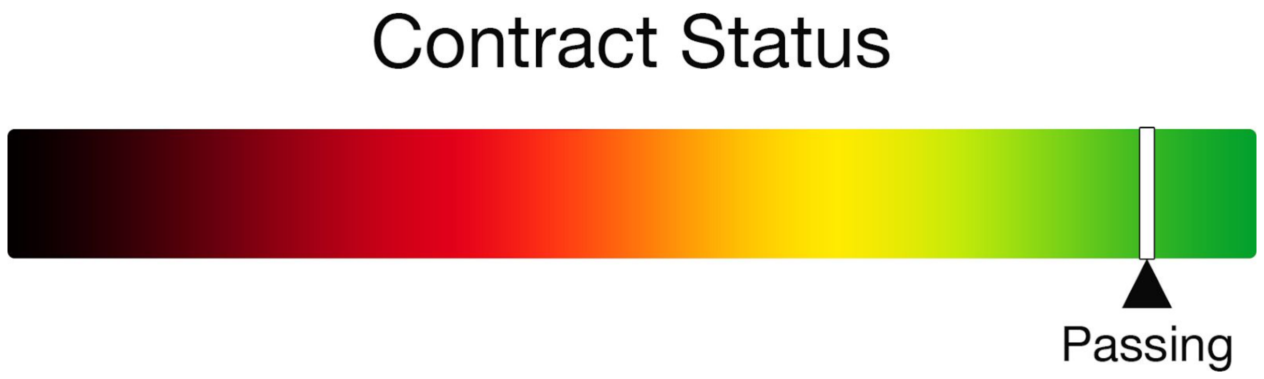
FanChain Contract Audit

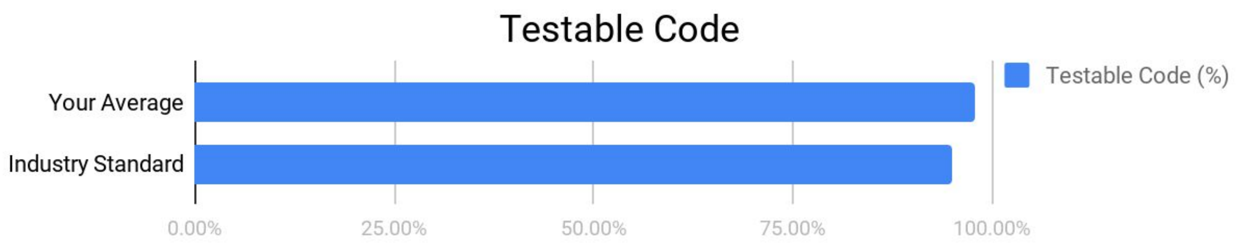
by Hosho, May 2018

Executive Summary

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



All issues have been remediated and suggestions implemented. (See Complete Analysis)



Testable code is 97.92% 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 FanChain 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 May 3, 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; and
- 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 FanChain's token contract. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as they were discovered. Part of this work included 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.

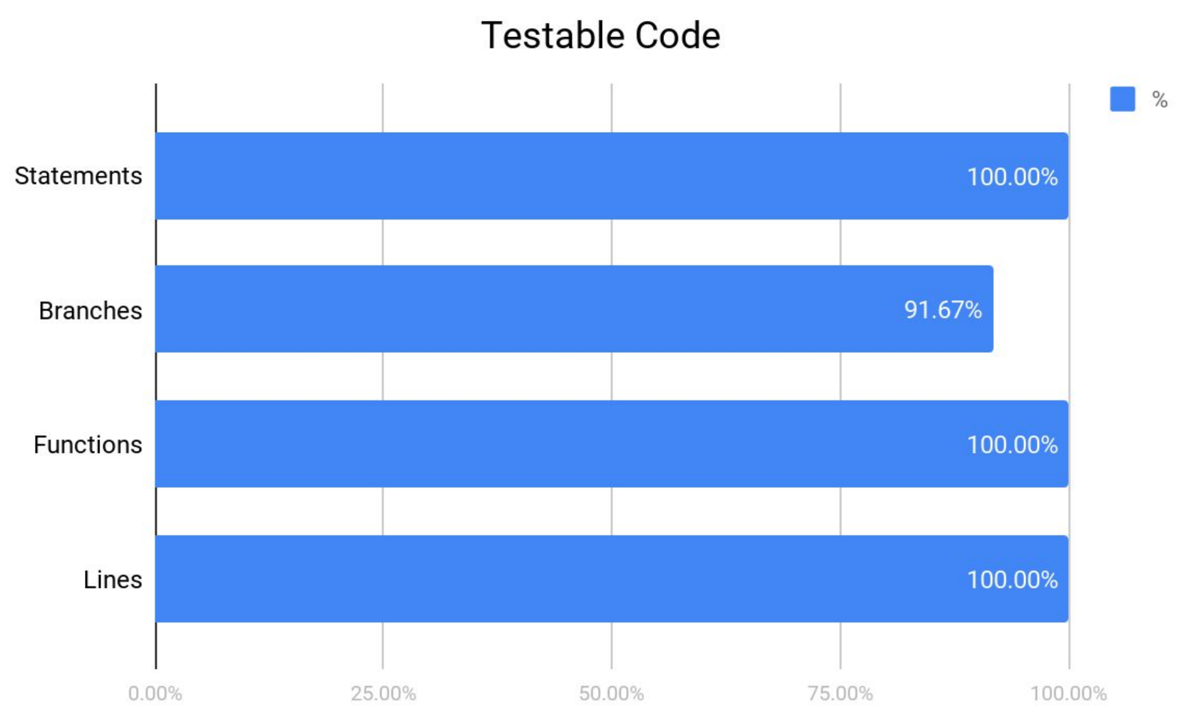
2. Structure Analysis and Test Results

2.1. Summary

FanCoin is an ERC-20 based token with a series of extensions that permit the "stamping" of coins to different "stamps". This allows the functional "transfer" of tokens between different systems within a single ERC-20 contract.

2.2 Coverage Report

As part of our work assisting FanChain 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:

- **Informational** - The issue has no impact on the contract's ability to operate.
- **Low** - The issue has minimal impact on the contract's ability to operate.
- **Medium** - The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.
- **High** - The issue affects the ability of the contract to compile or operate in a significant way.
- **Critical** - The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

1.1. Resolved, Critical: Transfer Not Functional

Contract: FanCoin

Explanation

The `transferFrom` function, in the FanCoin contract, calls `transferAny` after performing checks and then casts a value to be used as the transfer value. The `transferAny` function relies on `msg.sender,` instead of the authorized address that the tokens would be sent from: `address _from`. This creates the case that the contract attempts to send the tokens from whoever is executing the external call rather than the account that was authorized to send tokens.

Resolution

The FanChain Team updated the `transferAny` to use `_from` rather than `msg.sender` which resolves this issue.

1.2. Resolved, High: Stamp Override

Contract: Stampable

Explanation

The `stampToken` function allows an account that has been added to the stamping whitelist to apply any token ID to any token in any account. This leads to the potential for revenue loss and incorrect accounting within the system.

Technical Example

Assume the WNBA and the NFL both purchase one million tokens each and are stamping whitelisted. The WNBA stamps 1000 of their tokens and sends them to Jim Bob. The NFL can then stamp Jim's tokens in his account with any id that they choose as well as any remaining tokens in the WNBA's account, or anyone else's account for that matter. As long as an account is stamping whitelisted, any of the scenarios are possible.

Resolution

The `owner` parameter has been replaced with `msg.sender` by the FanChain Team, to specify whose balance the tokens are stamped from, which resolves this issue.

1.3. Resolved, Medium: ERC-20 Compliance

Contract: FanCoin

Explanation

The `transferToken` function emits a TokenTransfer event whenever it is called directly, or by other functions, that utilize its transfer capabilities. However, it does not emit the standard Transfer event, which will cause 3rd party integrations, such as Etherscan, to misreport transferred tokens.

Resolution

The standard Transfer event has been added by the FanChain Team, resolving this issue.

1.4. Resolved, Low: Missing Checks

Contract: FanCoin

Explanation

The function `transferTokens` receives an address, a list of tokens to transfer based on token ID and the the amounts to transfer, which are then executed as individual `transferToken` requests. However, there are no checks that either the IDs, or amounts, are real values, which allows the function to send an array of null values for both the token ids and the associated amounts. The result of this would be `transferToken` creating multiple calls with null values, leading to reporting issues and wasted gas.

Resolution

The FanChain Team has added validation to require non-zero transfer amounts, as well as checking for valid token balances, which fixes this issue.

1.5. Resolved, Informational: Hardcoded Owner Address

Contract: FanCoin

Explanation

We are unable to test with a hardcoded owner address and thus the owner parameter was changed to msg.sender for testing.

Resolution

The FanChain Team have been notified and acknowledge this.

---

4. Closing Statement

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

The team of experts at Hosho, having backgrounds in all aspects of blockchain, cryptography, and cybersecurity, can say with confidence that the FanChain 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 FanChain Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

5. Test Suite Results

Contract: ERC-20 Tests for FanCoin

√ Should deploy a token with the proper configuration (154ms)

√ Should allocate tokens per the minting function, and validate balances (2363ms)

√ Should `transfer` tokens from 0xd86543882b609b1791d39e77f0efc748dfff7dff to 0x42adbad92ed3e86db13e4f6380223f36df9980ef (832ms)

√ Should not `transfer` negative token amounts (111ms)

√ Should not `transfer` more tokens than you have (114ms)

√ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens (215ms)

√ Should require no existing allowed balance to approve (212ms)

√ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization (276ms)

√ Should allow 0x667632a620d245b062c0c83c9749c9bfadf84e3b to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and 0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to 0x341106cb00828c87cd3ac0de55eda7255e04933f (1190ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to `transfer` negative tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (95ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to `transfer` tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b to 0x0 (67ms)

√ Should not `transfer` tokens to 0x0 (80ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to `transfer` more tokens than authorized from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (123ms)

√ Should allow an `approval` to be set, then increased, and decreased (799ms)

Ensure 'FanCoin' defines the ERC20 Token Standard Interface

√ Should have the correct 'name' definition

√ Should have the correct 'approve' definition

```
    √ Should have the correct 'totalSupply' definition

    √ Should have the correct 'transferFrom' definition

    √ Should have the correct 'decimals' definition

    √ Should have the correct 'balanceOf' definition

    √ Should have the correct 'symbol' definition

    √ Should have the correct 'transfer' definition

    √ Should have the correct 'allowance' definition

    √ Should have the correct 'Transfer' definition

    √ Should have the correct 'Approval' definition


  Contract: More tests for FanCoin

    √ Should deploy a token with the proper configuration (131ms)

    √ Should allocate tokens per the minting function, and validate balances (2292ms)

    √ Should add 0xe05c3729d2380cbb23f3a2c2bac7c39a1e8357dd to the whitelist (123ms)

    √ Should fail to add a wallet to the whitelist from a non-owner account (85ms)

    √ Should remove 0xe05c3729d2380cbb23f3a2c2bac7c39a1e8357dd to the whitelist (120ms)

    √ Should stamp some tokens (1371ms)

    √ Should fail to stamp more tokens than the account has (940ms)

    √ Should fail to stamp more tokens from a non-whitelisted account (79ms)

    √ Should stamp some more tokens, but not add it to it's tokenBalances (1074ms)

    √ Should return the balance of all tokens 0x3b917236e9497521a4977ce09fe83fef7e01b4e0
has (53ms)

    √ Should return the balance of token with id 1 that
0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9 has (59ms)

    √ Should return which tokens 0x3b917236e9497521a4977ce09fe83fef7e01b4e0 has (65ms)

    √ Should transfer tokens of id 1 from 0x3b917236e9497521a4977ce09fe83fef7e01b4e0 to
0xa60d7faa9e9ce96fd434ed972812fcb080fc776c (597ms)

    √ Should fail to transfer tokens of id 1 from
0x3b917236e9497521a4977ce09fe83fef7e01b4e0 to 0x0 (80ms)

    √ Should fail to transfer more tokens of id 1 from
0x3b917236e9497521a4977ce09fe83fef7e01b4e0 than it has (104ms)
```

√ Should `transfer` tokens of id 1 and id 2 from 0x3b917236e9497521a4977ce09fe83fef7e01b4e0 to 0xa60d7faa9e9ce96fd434ed972812fcb080fc776c (1329ms)

√ Should fail to `transfer` tokens of id 1 and id 2 from 0x3b917236e9497521a4977ce09fe83fef7e01b4e0 to 0x0 (92ms)

√ Should fail to `transfer` tokens from 0x3b917236e9497521a4977ce09fe83fef7e01b4e0 to 0xa60d7faa9e9ce96fd434ed972812fcb080fc776c when given a list of values less than the list of ids (140ms)

√ Should fail to `transfer` tokens from 0x3b917236e9497521a4977ce09fe83fef7e01b4e0 to 0xa60d7faa9e9ce96fd434ed972812fcb080fc776c when given a list of ids and values larger than 100 items (146ms)

√ Should fail to transfer more tokens of id 1 and id 2 from 0x3b917236e9497521a4977ce09fe83fef7e01b4e0 to 0xa60d7faa9e9ce96fd434ed972812fcb080fc776c than it has (285ms)

√ Should issue tokens, stamp all of them, and remove the token id 0 from the account (1602ms)

√ Should issue tokens, stamp half as token 50, transfer 75% of them, and remove the token id 0 from the account (2721ms)

√ Should `mint` and `transfer` tokens (1621ms)

√ Should require valid recipient address for `mintTransfer` (110ms)

√ Should require enough balance to send in `mintTransfer` (137ms)

√ Should require all values in a `transferTokens` to be greater than zero (297ms)

Contract: SafeMath

√ Should revert on subtraction overflow (51ms)

√ Should allow regular subtraction (74ms)

√ Should revert on addition overflow (121ms)

√ Should allow regular addition (103ms)

6. All Contract Files Tested

Commit Hash: 5517842694798476eb7f9df8b366ce03da465130

File	Fingerprint (SHA256)
contracts/ERC20.sol	9c6f317cc2f898d50e46af982eabaea6fa61b2234128fe253a41989596255bb2
contracts/FanCoin.sol	6a4d9da1eac2725f7e0a6e94882545a5321b1127ffe73077036dfd41b85562d4
contracts/SafeMath.sol	081afba5a273466caa89c1453bd5530f07e9d6215fdb07ee7f72ef7f471b9581
contracts/Stampable.sol	7304f690ba013b66721c2f378b4cdba8eefd27494a0d7531bd8810728fb83e6f

7. Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
contracts/ERC20.sol	100.00%	100.00%	100.00%	100.00%
contracts/FanCoin.sol	100.00%	89.58%	100.00%	100.00%
contracts/SafeMath.sol	100.00%	100.00%	100.00%	100.00%
contracts/Stampable.sol	100.00%	100.00%	100.00%	100.00%
All files	**100.00%**	**91.67%**	**100.00%**	**100.00%**