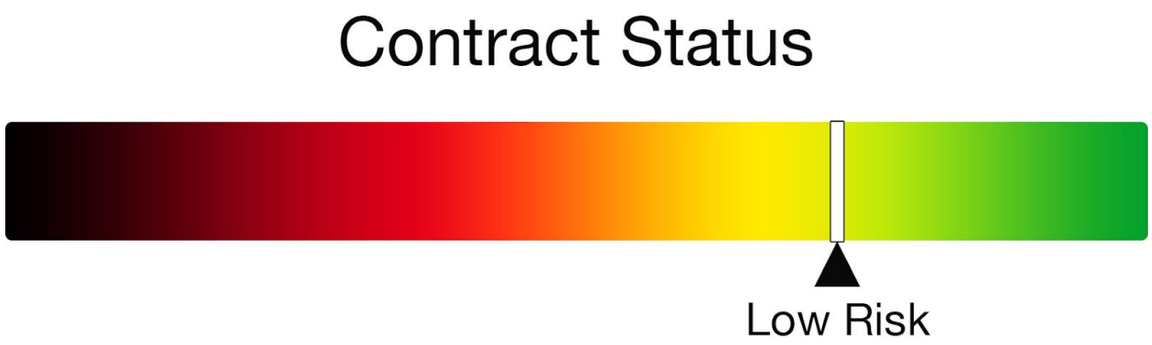
FirstBlood Contract Audit

Prepared by Hosho
October 2nd, 2018

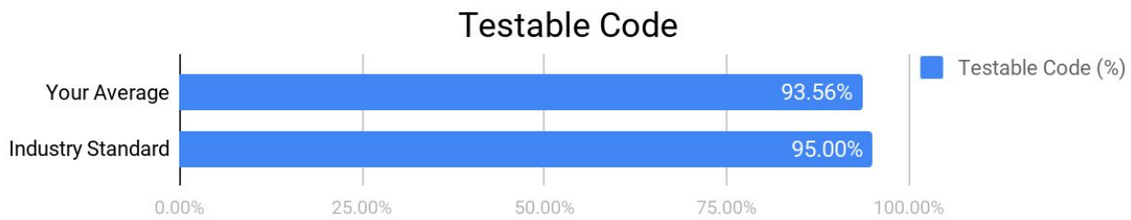
Report Version: 3.2

Executive Summary

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



There is a single low severity issue, that the FirstBlood team has acknowledged, and any remaining issues have been promptly rectified, rendering these contracts free from critical errors. (See Complete Analysis)



Testable code is 93.56% which is on par with the industry standard of 95%. (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 FirstBlood 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 July 10th, 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 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 FirstBlood's 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.

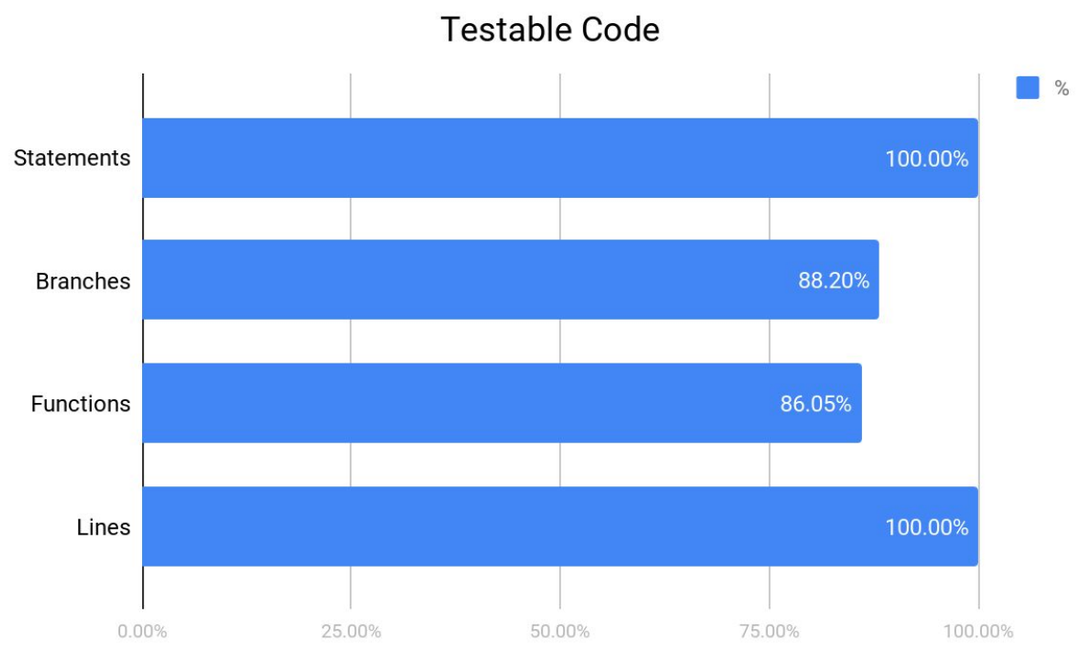
2. Structure Analysis and Test Results

2.1. Summary

The FirstBlood contracts make up a challenge and oracle receiver system, in which the FirstBlood team has implemented a sane way to handle wins and losses. The ability to confirm these is granted by the use of 3rd parties, as the oracles, to ensure that tokens are awarded properly. The FirstBlood team has made changes to their codebase to fix two previously found critical severity issues. Only one issue of low severity remains, but the FirstBlood team is aware of this and is expected functionality within their contracts.

2.2 Coverage Report

As part of our work assisting FirstBlood 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

1. Contract: ERC-20 Tests for ReserveToken - Should not transfer tokens to *0x0* (See Issue 3.6 Transfer to 0x0)
2. Contract: ERC-20 Tests for ReserveToken - Should not allow *0x53353ef6da4bbb18d242b53a17f7a976265878d5* to transfer tokens from *0x667632a620d245b062c0c83c9749c9bfadf84e3b* to *0x0* (See Issue 3.6 Transfer to 0x0)

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.

3.1 Resolved, Critical: Double Report Possible

Contract: WitnessJury

Explanation

One witness can call the `report` function twice, blocking the second witness from reporting and allowing them to receive both of the witness rewards.

Resolution

The FirstBlood team is now checking to ensure the *msg.sender* for the second witness is not equal to the first witness.

3.2 Resolved, Critical: Inconsistent Witness Results

Contract: WitnessJury

Explanation

The `resolve` function in this contract relies on the reported winners to be the same in order to execute. If witness 1 and witness 2 report a different winner for any reason, even if a jury votes on the reported results, this contract never calls the Challenge's `resolve` function. This leads to the potential winnings to be locked in the contract, unable to be escaped.

Resolution

The FirstBlood team is no longer requiring that *winner1* and *winner2* agree, and if they do not, a jury is still able to vote.

3.3 Resolved, High: Minimum Bet Too Low

Contract: Challenge

Explanation

Due to the setup of the equations for calculating the amounts to transfer in the resolve function, any bet under 40 causes the contract to break on token `transfer`. Currently, the constructor for this contract only limits the bet amount to greater than or equal to zero. The math either needs to be updated to accommodate smaller bets, or the constructor updated to set the minimum bet that works with these equations.

Resolution

The FirstBlood team has implemented conditional logic to validate they are not sending more tokens than are available.

3.4 Resolved, High: Authorization Issue in transferFrom

Contract: Challenge

Explanation

The public `respond` function takes the *address* of the responding user as a *parameter* instead of handling it as *msg.sender*, and then calls `transferFrom` from that *address*. This means that Bob is able to respond to a challenge using Alice's *address* instead of his own. The contract will pull funds from Alice's account instead of Bob's, without any protection for Alice, other than the `allowance` amount for Alice’s account.

Resolution

The FirstBlood team is checking that the responding user is now the sender of the message.

3.5 Resolved, Low: Transfer Without Funds

Contract: Challenge

Explanation

If user1 for this contract calls the `rescue` function after the block period but before it is funded, the contract will still attempt to `transfer` the amount to user1 and *revert* even though the contract has not actually been sent any tokens.

Resolution

The FirstBlood team is now completing a check that requires the contract is funded prior to execution.

3.6 Unresolved, Low: Transfer to 0x0

Contracts: Token, StandardToken

Explanation

There are currently no protections against the `transfer` or `transferFrom` functions sending tokens to the contract address *0x0*.

Update

The FirstBlood team has acknowledged this functionality and as they require the ability to destroy tokens from any account and don't need to block transfers to address *0x0,* this is by design.

---

4. Closing Statement

The Hosho team is grateful to have been given the opportunity to work with the FirstBlood team.

The FirstBlood contracts are a series of contracts designed with the goal of providing an oracle-based gaming tournament system. This allows for the automation of tournaments, payouts, and win/loss tracking on the blockchain. The FirstBlood team have fixed two issues of critical severity in the WitnessJury contract as well as several other issues of lower severity.

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 FirstBlood 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

Contract: ERC-20 Tests for ReserveToken

√ Should deploy a token with the proper configuration (87ms)

√ Should allocate tokens per the minting function, and validate balances (314ms)

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

√ Should not transfer negative token amounts | (67ms)

√ Should not transfer more tokens than you have (62ms)

√ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens (60ms)

√ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization (60ms)

√ Should allow 0x667632a620d245b062c0c83c9749c9bfadf84e3b to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and 0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to 0x341106cb00828c87cd3ac0de55eda7255e04933f (184ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer negative tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (63ms)

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

X Should not transfer tokens to 0x0

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer more tokens than authorized from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (72ms)

Contract: Usage tests for ReserveToken

√ Should deploy a token with the proper configuration (52ms)

√ Should allocate tokens per the minting function, and validate balances (255ms)

√ Should fail to create tokens from a non-owner

√ Should fail to destroy tokens from non-owner (38ms)

√ Should fail to destroy more tokens than a user has

√ Should destroy tokens (72ms)

Contract: SafeMath

√ Should skip operation on multiply by zero

√ Should revert on multiply overflow

√ Should allow regular multiple

√ Should revert on subtraction overflow

√ Should allow regular subtraction

√ Should revert on addition overflow

√ Should allow regular addition

Contract: Firstblood tests

Challenge tests

√ Should not construct with a value of 0 (80ms)

Fund, respond, and host tests

√ Should fund, respond, and host a challenge (270ms)

Fund tests

√ Should fail to fund a challenge with an invalid transfer (88ms)

√ Should fail to fund a challenge that has already been funded (117ms)

√ Should fail to fund a challenge from the wrong user (42ms)

√ Should fund a challenge, but not let any user just send the address of the another responding account.

Respond tests

√ Should fail to respond to a challenge that hasn't been funded

√ Should fail to respond to a challenge with an invalid transfer (159ms)

√ Should fail to respond to a challenge that has already been responded to (219ms)

√ Should fail to respond to a challenge with user2 address 0

√ Should fail to respond to a challenge that has been rescued (876ms)

Host tests

√ Should fail to host a challenge that has already been hosted (84ms)

√ Should fail to host a challenge that has been rescued (933ms)

setWitnessJury tests

√ Should `setWitnessJuryKey` (247ms)

√ Should fail to `setWitnessJuryKey` twice (272ms)

√ Should fail to `setWitnessJuryKey` after rescue (1108ms)

√ Should fail to `setWitnessJuryKey` from non-host (85ms)

Rescue tests

√ Should fail to rescue a challenge that has already been rescued (865ms)

√ Should fail to rescue a challenge before block time (115ms)

√ Should fail to rescue a challenge from an unauthorized user (795ms)

requestJury tests

√ Should request a jury (1102ms)

ChallengeFactory tests

√ Should create a new challenge (130ms)

WitnessJury tests

√ Should deposit (166ms)

√ Should deposit but not increase the number of witnesses (265ms)

√ Should fail to deposit with amount 0 (80ms)

√ Should fail to deposit with bad token transfer (75ms)

√ Should withdraw some tokens (248ms)

√ Should withdraw all tokens (276ms)

√ Should fail to withdraw 0 tokens (181ms)

√ Should fail to withdraw more tokens than deposited (329ms)

√ Should fail to withdraw tokens with failing transfer (274ms)

√ Should check `isJuror` (189ms)

√ Should `reduceToLimit` (44ms)

√ Should fail to `reduceToLimit` from a different address than passed

√ Should `reduceToLimit` with greater than limit (309ms)

√ Should fail to `reduceToLimit` with greater than limit and failing transfer (331ms)

√ Should get `balanceOf()`

√ Should `getRequest()`

√ Should make a new request over the `DRMVolumeCap()` (119ms)

√ Should fail to check `juryNeeded` from non challenge account

Contract: Resolve and Jury tests

√ Should resolve (582ms)

√ Should fail to resolve a challenge that already has a winner (631ms)

√ Should fail to resolve a challenge when sent a wrong `witnessJuryRequestNum` (329ms)

√ Should fail to resolve a challenge from an address that isn't the `witnessJury` set for the Challenge contract (300ms)

√ Should fail to resolve a challenge with an invalid winner (324ms)

√ Should fail to resolve a challenge that has been rescued (1376ms)

√ Should fail to resolve a challenge before the block period has passed (310ms)

√ Should resolve a challenge with winner 1 (600ms)

√ Should resolve a challenge with winner 2 (564ms)

√ Should resolve a challenge with a contested Jury (1564ms)

√ Should fail to request a jury from the wrong account (308ms)

√ Should fail to report from non-jury (52ms)

√ Should fail to report from the same account twice

√ Should fail to report a third time (301ms)

√ Should fail to request after rescue (1431ms)

√ Should fail to `requestJury` after winner is decided (687ms)

√ Should fail to request a jury twice (392ms)

√ Should fail to request a jury before both witness reports (224ms)

√ Should fail to resolve a challenge with a failing transfer to winner 1 (525ms)

√ Should fail to resolve a challenge with a failing transfer to winner 2 (561ms)

√ Should resolve a challenge with a referrer (1542ms)

√ Should fail to resolve a challenge with a failing transfer to host (723ms)

√ Should fail to resolve a challenge with a failing transfer to Witness1 (510ms)

√ Should fail to resolve a challenge with a failing transfer to Witness2 (512ms)

√ Should resolve a challenge with excess balance (663ms)

√ Should fail to rescue a challenge that has already been resolved (1590ms)

6. Appendix B

All Contract Files Tested

Commit Hash: 916d067b119f861f9938c029198d7f4d467db828

File	Fingerprint (SHA256)
Compiled.sol	`3f0d06d521c1d1c86bf1a4cb331c809b2c4c4721a8a4fe9bba38616aaedde2aa`

7. Appendix C

Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
Compiled.sol	100.00%	88.20%	86.05%	100.00%
All files	**100.00%**	**88.20%**	**86.05%**	**100.00%**