Crypto20 Token Contract Audit

by Hosho, October 2017

Table of Contents

Technical Summary

This document outlines the overall security of Crypto20's smart contract as evaluated by Hosho's Smart Contract auditing team. The scope of this audit was to analyze and document Crypto20's token contract codebase for quality, security, and correctness.

The Crypto20 contract is an ERC-20 token with some heavy modifications made to support their unique platform for ability to sell tokens back through the contract for ETH at any time after the crowdsale has ended.

The contract is well written and shows good performance under dynamic analysis, with a solid basis in the ERC-20 standard. Conversion from dynamic time allocations to static block heights helps to ensure that tokens are issued at the correct rates and as expected, and follow the issuance guidance from the WhitePaper. Due to the complexity of the fallback function, thanks to how issuance is handled, the contract is not able to accept transactions from other contracts (most likely, Multisig wallets from more advanced investors), however, equivalent entry points can be found in the `buy` and `buyTo` functions.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, 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 Crypto20 Team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Auditing Strategy and Techniques Applied

The Hosho Team has performed a thorough review of the smart contract code as written and provided on October 3, 2017. The following contract files and their respective SHA256 fingerprints were evaluated:

File	Fingerprint (SHA256)
C20.sol	`7ba25dbd0cb5776d6cd8ec8bae950e3a04ad22aa61f78e300b12dd6023292f91`
C20Vesting.sol	`058f4b828b3e5c2c00198b78a445725516abda76068f0ddab25c94f6fa733188`
SafeMath.sol	`3f5194fb92f4eb260776b2592deb36e27f8ea5a9d5cb09b849b14939ec078e28`
StandardToken.sol	`b4aa3f758356a30a9cd12e20fb687ae3ad8de19080030a195b9c82e4581fa5e6`
Token.sol	`e133dd392b81191d49bb3458ae97d9c543252c2dcf5d989d82237fe6f237b316`

A second follow-up review was done on the updated code from October 10, 2017.

File	Fingerprint (SHA256)
C20.sol	`0b78611ba10106e70bb6a7d250ae6b39bdb6096c0f51f51c2ab084f3b5008b1b`

C20Vesting.sol	`4bce7d31fb8edf24b2340faf25a8a3f671564b7f6e359f968222bf241a11deac`
SafeMath.sol	`3f5194fb92f4eb260776b2592deb36e27f8ea5a9d5cb09b849b14939ec078e28`
StandardToken.sol	`b4aa3f758356a30a9cd12e20fb687ae3ad8de19080030a195b9c82e4581fa5e6`
Token.sol	`e133dd392b81191d49bb3458ae97d9c543252c2dcf5d989d82237fe6f237b316`

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

- Implements and adheres to existing ERC20 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; and
- Uses methods safe from reentrance attacks.

The Hosho Team has followed best practices and industry-standard techniques to verify the implementation of Crypto20'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.

Contract Analysis and Test Results

Summary

The Crypto20 token is a compliant ERC-20 Token with additional functionality added to adhere to the rules structured for their crowdsale.

During the audit several critical issues were discovered that were quickly resolved. The contract is now ready for deployment.

Coverage Report

As part of our work assisting Crypto20 in verifying the correctness of their contract code, our team was responsible for writing a unit test suite using the Truffle testing framework.
Non-covered branches are largely due to inaccessible locations, such as a `this.balance`

less-than-or-equal when `this.balance` has already been confirmed as greater than the value. Other such locations are `requires` as contained within the initialization functions.

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

File	% Statements	% Branches	% Functions	% Lines
C20.sol	100.00%	91.67%	100.00%	100.00%
C20Vesting.sol	100.00%	96.88%	100.00%	100.00%
SafeMath.sol	100.00%	50.00%	100.00%	100.00%
StandardToken.sol	100.00%	100.00%	100.00%	100.00%
Token.sol	100.00%	100.00%	100.00%	100.00%
All files	100.00%	91.22%	100.00%	100.00%

Test Results

Contract: ERC-20 Compliant Token

- ✓ Should deploy with Crypto20 as the name of the token (66ms);
- ✓ Should deploy with C20 as the symbol of the token (76ms);
- ✓ Should deploy with 18 decimals;
- ✓ Should deploy with 0 tokens;
- ✓ Should deploy with `fundWallet` set to 0x7b11c8d4870c4c612f528d7b45a7fab18645572f;
- ✓ Should deploy with `controlWallet` set to 0xab5c4a8c060bbb46128dfbb928a5233a5d14cd27 (40ms);
- ✓ Should deploy with `fundWallet` and `controlWallet` in whitelist (79ms);
- ✓ Should deploy with `currentPrice` numerator set to 1 and denominator set to 1000 (48ms);
- ✓ Should deploy with `fundingStartBlock` set to 100000 (40ms);
- ✓ Should deploy with `fundingEndBlock` set to 1000000 (39ms);
- ✓ Should deploy with non-zero `previousUpdateTime`;
- ✓ Should deploy with `waitTime` set to 5 hours;
- ✓ Should deploy with `halted` set to false;
- ✓ Should deploy with `tradeable` set to false (39ms);
- ✓ Should deploy with `minAmount` set to 0.04 ether (39ms);
- ✓ Should not allow Vesting Contract to be set to a null address (47ms);
- ✓ Should not allow `updatePrice` to proceed before the `waitTime` has elapsed if called by `controlWallet` (118ms);
- ✓ Should allow the price to be updated by the `fundWallet` at any time. (103ms);

- ✓ Should not allow `updatePrice` to be called by the `fundWallet` as it should be blocked by `require_limited_change` if they don't match (63ms);
- ✓ Should update the `currentPrice` numerator with 20% limit if called by `controlWallet` (428ms);
- ✓ Should not allow the `controlWallet` to reduce the value of the numerator (331ms);
- ✓ Should not update the `currentPrice` numerator with 20% limit if called by `controlWallet` (348ms);
- ✓ Should allow the price to be updated by the `fundWallet` at any time. (77ms);
- ✓ Should not let you set the `updateFundingStartBlock` to less than the current block (45ms);
- ✓ Should not let you set the `updateFundingEndBlock` to less than the current block (45ms);
- ✓ Should not let you enable trading before the `endBlock` is hit;
- ✓ Should not allow Vesting Contract to be set by non-`fundWallet` account;
- ✓ Should set Vesting Contract to `Vesting.address` and add `Vesting.address` to `whitelist` (115ms);
- ✓ Should not allow `updatePrice` to be called by non-managing wallets (38ms);
- ✓ Should not allow `updatePrice` to set a price of 0;
- ✓ Should not allow `updatePriceDenominator` to proceed before `fundingEndBlock` (48ms);
- ✓ Should allocate tokens per the minting function, and validate balances (1059ms);
- ✓ Should not transfer before trading is enabled (40ms);
- ✓ Should not allow `transferFrom` before trading is enabled};
- ✓ Should not allow `verifyParticipant` to be called by non-`managingWallets` (40ms);
- ✓ Should allow whitelisting of participants by `fundWallet` (78ms);
- ✓ Should not allow `requestWithdrawal` to be called before `fundingEndBlock` (60ms);
- ✓ Should not allow `buyTo` function to be called before `fundingStartBlock` (60ms);
- ✓ Should allow accounts to purchase tokens within `fundingStartBlock`-`fundingEndBlock` window (1051ms);
- ✓ Should not allow `buyTo` function to be called by accounts not in the whitelist;
- ✓ Should not allow the buy function to be called when halted (83ms);
- ✓ Should not allow the `buyTo` function to be called to an address of 0 (97ms);
- ✓ Should not allow the buy function to submit funds under the `minAmount` (102ms);
- ✓ Should throw an error if `checkWithdrawValue` is not a valid amount (60ms);
- ✓ Should not let you set the `fundWallet` to 0;
- ✓ Should not let you set the `controlWallet` to 0 (38ms);
- ✓ Should not allow `buyTo` function to be called by when contract is halted (150ms);

- ✓ Should allow fallback function to buy tokens within `fundingStartBlock-fundingEndBlock` window (252ms);
- ✓ Should return ico denominator values depending on time since previous update (62671ms);
- ✓ Should update the `currentPrice` denominator (630ms);
- ✓ Should not allow `updatePriceDenominator` to proceed with `newDenominator` of 0 (46ms);
- ✓ Should not allow `updatePriceDenominator` to be called by non-`fundWallet` (41ms);
- ✓ Should not allow `requestWithdrawal` to be called when `isTradeable` is false;
- ✓ Should not allow `requestWithdrawal` to be called when by non-whitelisted account;
- ✓ Should allow a withdrawal request and withdraw execution but fail to return ether (521ms);
- ✓ Should not allow `requestWithdrawal` to be called for `amountToken` equal to 0 (52ms);
- ✓ Should not allow `requestWithdrawal` to be called for `amountToken` greater than account balance (73ms);
- ✓ Should not allow withdraw to be called with a price mapping `previousUpdateTime` to zero (252ms);
- ✓ Should check the withdrawal amount value (83ms);
- ✓ Should not allow `checkWithdrawValue` to be called for `amountToken` equal to 0;
- ✓ Should not allow `checkWithdrawValue` to be called for a balance greater than stored on the contract (81ms);
- ✓ Should not allow a null amount of liquidity to be added to the contract;
- ✓ Should not allow `addLiquidity` to be called by non managing wallets;
- ✓ Should allow a withdrawal request and withdraw execution and successfully return ether (446ms);
- ✓ Should not allow a withdrawal when the user hasn't requested one;
- ✓ Should not allow presale tokens to be allocated after the end of funding;
- ✓ Should not allow `requestWithdrawal` to be called for accounts with outstanding withdrawals (212ms);
- ✓ Should not allow `removeLiquidity` to be called by non managing wallets;
- ✓ Should not allow `removeLiquidity` to transfer a balance greater than stored on the contract;
- ✓ Should decrease the ether balance of the `C20` contract (261ms);
- ✓ Should not allow `fundWallet` to be changed by non-`fundWallet` account;
- ✓ Should change the fund wallet to accounts[2] (73ms);
- ✓ Should not allow `controlWallet` to be changed by non-`fundWallet` account (40ms);

- ✓ Should change the control wallet to accounts[2] (67ms);
- ✓ Should not allow `changeWaitTime` to be called by non-`fundWallet` account;
- ✓ Should change the wait time to 1 hour (74ms);
- ✓ Should not allow `halt` to be called by non-`fundWallet` account;
- ✓ Should set `halted` to true (72ms);
- ✓ Should not allow `unhalt` to be called by non-`fundWallet` account;
- ✓ Should set halted to false (62ms);
- ✓ Should not allow `enableTrading` to be called by non-`fundWallet` account;
- ✓ Should set `tradeable` to true (78ms);
- ✓ Should not allow `buyTo` function to be called after `fundingEndBlock` (99ms);
- ✓ Should transfer tokens from 0x7b11c8d4870c4c612f528d7b45a7fab18645572f to 0xc7ef17e5ba85c1eeb707529a2adeda7612d1b9ff (164ms);
- ✓ Should not transfer negative token amounts (58ms);
- ✓ Should not transfer tokens to the void (58ms);
- ✓ Should not transfer more tokens than you have (73ms);
- ✓ Should allow 0x36cefd837af98d93ad83e2e9b8bc68757cf7ef6e to authorize 0xc7ef17e5ba85c1eeb707529a2adeda7612d1b9ff to transfer 1000 tokens (110ms);
- ✓ Should not allow 0x36cefd837af98d93ad83e2e9b8bc68757cf7ef6e to authorize 0xc7ef17e5ba85c1eeb707529a2adeda7612d1b9ff to transfer an additional 1000 tokens once authorized, and `authorization` balance is > 0 (52ms);
- ✓ Should allow 0x36cefd837af98d93ad83e2e9b8bc68757cf7ef6e to authorize 0xc7ef17e5ba85c1eeb707529a2adeda7612d1b9ff to transfer an additional 1000 tokens once authorized, and `authorization` balance is > 0 if done through `changeApproval` (135ms);
- ✓ Should not allow 0x36cefd837af98d93ad83e2e9b8bc68757cf7ef6e to authorize 0xc7ef17e5ba85c1eeb707529a2adeda7612d1b9ff to change the approval if the old value doesn't match (43ms);
- ✓ Should allow 0x36cefd837af98d93ad83e2e9b8bc68757cf7ef6e to zero out the 0xc7ef17e5ba85c1eeb707529a2adeda7612d1b9ff authorization (87ms);
- ✓ Should allow 0xab5c4a8c060bbb46128dfbb928a5233a5d14cd27 to authorize 0x1781a5e4dd4fb9718615ec70247f05922ab49b13 for 1000 token spend, and 0x1781a5e4dd4fb9718615ec70247f05922ab49b13 should be able to send these tokens to 0xc7ef17e5ba85c1eeb707529a2adeda7612d1b9ff (329ms);
- ✓ Should not allow 0x1781a5e4dd4fb9718615ec70247f05922ab49b13 to transfer negative tokens from 0xab5c4a8c060bbb46128dfbb928a5233a5d14cd27 (78ms);
- ✓ Should not allow 0x1781a5e4dd4fb9718615ec70247f05922ab49b13 to transfer tokens to the void (59ms);
- ✓ Should not let you set the `updateFundingStartBlock` after funding has started;

- ✓ Should not let you set the `updateFundingEndBlock` after funding has ended (46ms);
- ✓ Should transfer external tokens from itself to the `fundWallet` (116ms);
- ✓ Should only let the `fundWallet` retrieve tokens; and
- ✓ Should not allow a token address of 0 (38ms)

Contract: C20Vesting

- ✓ Should test that `C20Vesting` initialized with beneficiary set to 0x7b11c8d4870c4c612f528d7b45a7fab18645572f;
- ✓ Should test that `C20Vesting` initialized with `fundingEndBlock` 5000 (38ms);
- ✓ Should not let you set the `updateFundingEndBlock` to less than the current block (40ms);
- ✓ Should not let you set the `updateFundingEndBlock` if you're not the `beneficiary` (39ms);
- ✓ Should let you update the end block if you are the `beneficiary` (79ms);
- ✓ Should test that `ERC20Token` points to the `C20` contract address;
- ✓ Should test that claim can only be called after `fundingEndBlock`;
- ✓ Should test that `changeBeneficiary` changes variable `beneficiary` (116ms);
- ✓ Should test that `changeBeneficiary` can not set a null address (44ms);
- ✓ Should test that `changeBeneficary` can not be called from non-`beneficiary` account (40ms);
- ✓ Should not allow `allocateTokens` to be called before `vestingSet` equals true (62ms);
- ✓ Should allocate tokens per the minting function, and validate balances (832ms);
- ✓ Should not allow `allocateTokens` to use a value greater than `tokenCap` (134ms);
- ✓ Should not allow `allocatePresaleTokens` to proceed with a null participant address (52ms);
- ✓ Should test that `checkBalance` correctly returns the `ERC20Token` balance (55ms);
- ✓ Should test the initialized state of stage;
- ✓ Should test that claim can only be called by the `beneficiary`;
- ✓ Should test that claim initializes correctly when called after fundingEndBlock (1510ms);
- ✓ Should test release stages were set correctly after `init_claim` (91ms);
- ✓ Should test that stage incremented correctly after claim;
- ✓ Should not issue a `first_claim` prior to the first 26 weeks (122ms);
- ✓ Should test that claim proceeds to `first_release` after 26 weeks (569ms);
- ✓ Should test that stage incremented correctly after claim;
- ✓ Should not issue a `second_claim` prior to the second 26 weeks (89ms);
- ✓ Should test that claim proceeds to `second_release` after 26 weeks (515ms);
- ✓ Should test that stage incremented correctly after claim;

- ✓ Should not issue a `third_claim` prior to the third 26 weeks (84ms);
- ✓ Should test that claim proceeds to `third_release` after 26 weeks (513ms);
- ✓ Should test that stage incremented correctly after claim;
- ✓ Should not issue a `fourth_claim` prior to the fourth 26 weeks (76ms);
- ✓ Should test that claim proceeds to `fourth_release` after 26 weeks (627ms);
- ✓ Should test that stage incremented correctly after claim;
- ✓ Should transfer external tokens from itself and the token contract to the owner (108ms);
- ✓ Should only let the `beneficiary` retrieve tokens;
- ✓ Should not allow a token address of 0 (43ms);
- ✓ Should not allow a token address of the main sale token (65ms); and
- ✓ Should not let you set the `updateFundingEndBlock` if the end block is already hit (46ms)

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:

- **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.

Complete Analysis

Resolved, Critical: Logic Error Affecting ICO Duration

C20.sol:L215

Explanation

Before the audit began, the Hosho team provided information on a bug located within the `previousUpdateTime` handler, in regards to the `icoDenominatorPrice` function. This is based on the `previousUpdateTime`, and therefore, could be reset during the ICO itself

which would cause the 24 hour timer to reset. That bug in specific had been fixed by the CryptoTwenty team before the audit began, however, the `previousUpdateTime` is still being used as the starting point for the 24 hour offset for the contract. This leaves open a different bug where the contract may have the price updated well before the `fundingStartBlock` is hit. In this case, it means the 24 hour period is not from the `fundingStartBlock`, but instead, from the last time `previousUpdateTime` was set. If that value were to be last updated over 24 hours before the `fundingStartBlock`, then the contract will shift immediately into using the 4 week long duration.

Resolution

This was resolved by using a block number timing scheme for icoDenominatorPrice.

Resolved, Informational: Undesired Failure Modes in Function

C20.sol:L132-L143

Explanation

During negative assertion testing of the `updatePrice` function, it was noted that shifts > 20% are permitted, as well as shifts before the 5 hour time period is up. Due to the way that `require_limited_change` is being called as a function rather than a modifier and several core lockouts are written as modifiers, this function triggers an interesting issue. If any of the modifiers fail, they simply cause the code to not execute rather than causing the code to revert. The caller is expecting a revert so rather than checking for a return value, it allows the contract to continue without verifying the 5 hour time period lockout, the 20% shift max, or most importantly, that the value can only go up rather than down.

`require_limited_change function` utilizes the following modifiers:

```
only_if_controlWallet
Require_waited
only_if_increase(newNumerator)
```

When directly calling a solidity function, a modifier either allows the code to execute, or causes it to not execute, acting like a `revert()`, but only when called directly. This is due to the fact that a modifier's special `_;` value is replaced with the run-time code, but only as part of the modifier itself, which in all three cases, are simple if-statements. Due to this, when a modifier is called as part of an internal function, it simply blocks the code from running. As such, the code is swapped into the if block, which then causes it to be not-executed when the if-statement resolves to `false`.

In our test case, the `require_waited` function resolved to `false`, which caused the function to return without ever causing a `revert()`, thereby allowing the change both early, and to be over the 20% limit imposed by the function itself. This would also resolve `false` if the `numerator` that was new was less than the current `numerator`, which would cause the function to return and allow the amount to be lowered.

Resolution

It has been noted that `require_waited` does have a `require` within. Also, the Crypto20 team has explained that the remaining behavior is as expected. The `fundWallet` is permitted to change the value any time, so they have a hard-revert on the 5 hour time only on the automated wallet changes from `controlWallet`. Decrease amounts are unlimited. They can only limit a 20% rise in the numerator. This issue has been lowered to an Informational state.

Resolved, Medium: Unlinked Corresponding Values

C20Vesting.sol:L80

Explanation

In the `C20Vesting` contract, it is written to rely on a `fundingEndBlock` variable, which is set when the contract is deployed.

The issue with this is that the main `C20` contract can have its end block changed, but the `C20Vesting` can not. This normally won't present an issue, but it does mean the two contracts can have different `fundingEndBlock`s set and the latter contract uses this as a lockout for the `claim()` function. That function sets up the initial release of tokens, so the contract may end early, but you wouldn't be able to claim from this.

Resolution

This was resolved by linking the end block to a specific ID so that there cannot be an inconsistency between the `C20` contract and `C20Vesting`.

Resolved, Informational: Unclear Expectation

C20.sol:L233

Explanation

During the `requestWithdrawal` function, the `withdraw` statement is set to a time of `previousUpdateTime`. Then, during the `withdraw` execution itself this data is loaded from the `prices` data array. There is questionable behavior in regards to how this data array is initialized. Per the current code base, once the ICO is complete you can request the withdrawal at any time, however either the `numerator` or the `denominator` needs to be updated before `withdraw()` itself can be called. Otherwise, the price array is not valid at the `requestTime`, which will cause the system to not allow the withdrawal after this is changed.

As this behavior isn't immediately clear we thought it best that it be explained in case this execution is not designed as intended.

Resolution

This was designed as intended to enforce an SEC mandated policy of forward pricing withdrawals. Withdrawals will only be priced for the next value, not the existing one which prevents well-informed persons from withdrawing with knowledge that has occurred since the previous valuation was set.

Resolved, Informational: Suggested Augmentation

C20.sol

Explanation

The Hosho auditing team also suggests adding some type of token failsafe to the `C20` contract in the case that someone sends tokens to the contract. This is usually as simple as installing a very basic ERC-20 interface to the contracts themselves, then allowing the input of a contract address to a function that can call outbound to the other contract that provides the token, and calling the `transfer` function to move them to a normal wallet, usually an owner or a more-equipped to handle contract. This is not needed, but serves as a good way to handle sending tokens, rather than ETH to a sale contract or other contract as there is no blocking in ERC-20 to ensure that tokens aren't sent to places where they cannot be retrieved.

Resolution

The Crypto20 team added a failsafe to the `C20Vesting` contract to ensure that non-C20 tokens can be returned. There is a hard lock on the `claimOtherTokens` function to not allow `C20` tokens to be returned.

Closing Statement

It has been a pleasure to work with the Crypto20 team, who have been receptive to our feedback and made themselves available to us throughout the review process. The Crypto20 team's responsiveness and explanation to the reasoning behind some of the code's design was very helpful in the auditing process.

With recovery functions for external tokens, and smart locks provided onto the Vesting contract to ensure that the vested tokens can't be transferred early, the Crypto20 team is taking great care to help ensure that tokens can't be locked up accidentally within their contracts.

Overall, the contract is very functional, and passes on both dynamic and static reviews after the changes advised 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 Crypto20's contracts are free of any glaring issues and the team has shown itself to be capable of removing issues quickly and effectively when presented. 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 Crypto20 Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.