CoinAlpha Contract Audit

Prepared by Hosho
July 23rd, 2018

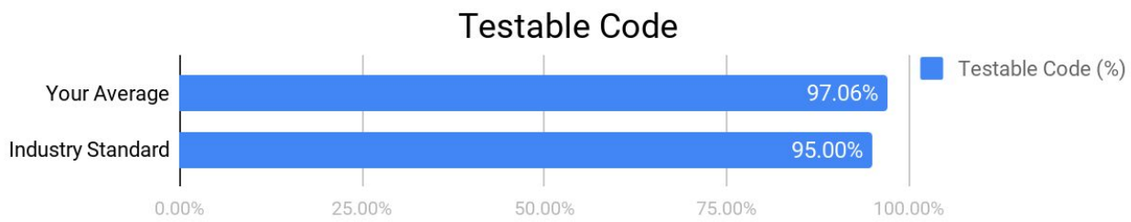
Report Version: 2.0

Copyright © 2018 Hosho Group Inc.

Executive Summary

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

These contracts have passed the rigorous auditing process performed by the Hosho team. (See Complete Analysis)



Testable code is 97.06% which is greater than 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 CoinAlpha 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 3

2. Structure Analysis and Test Results 4

2.1. Summary

2.2 Coverage Report

2.3 Failing Tests

3. Complete Analysis 5

3.1 Resolved, Medium: Use of Assert Instead of Require

3.2 Resolved, Medium: No Time Validation on Expiration

3.3 Resolved, Low: No Overflow Protection

3.4 Resolved, Low: No Overflow Protection

4. Closing Statement 7

5. Appendix A 8

Test Suite Results

6. Appendix B 14

All Contract Files Tested

7. Appendix C 15

Individual File Coverage Report

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 16th, 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 CoinAlpha'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.

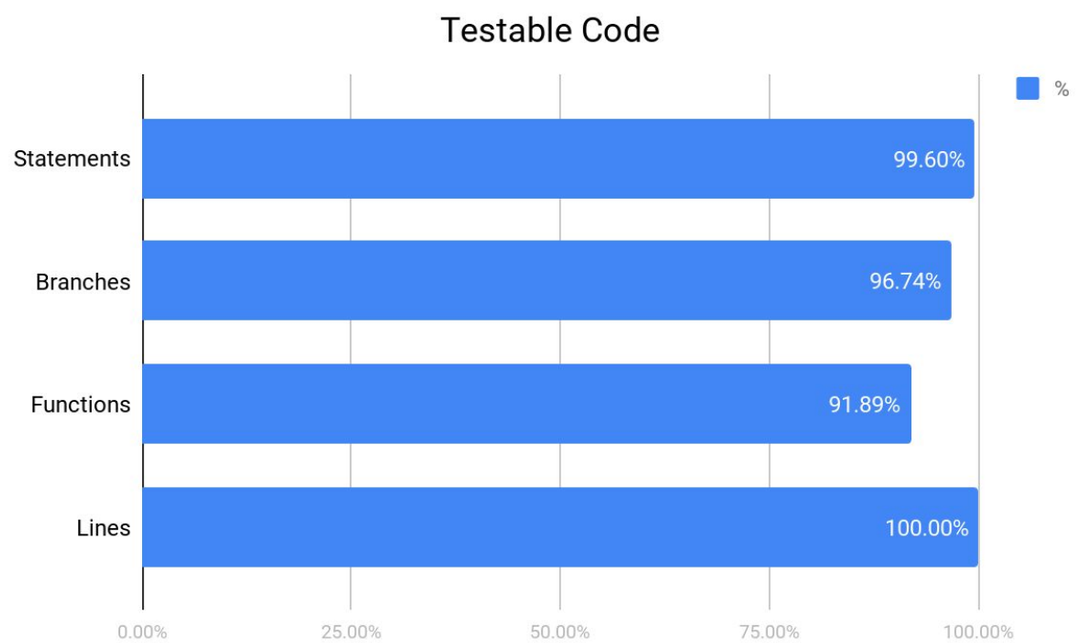
2. Structure Analysis and Test Results

2.1. Summary

The CoinAlpha contracts establish a decentralized investment system that utilizes a Basket to collect, bundle, and invest different tokens and assets as a non-custodial portfolio, over which the investor has full control.

2.2 Coverage Report

As part of our work assisting CoinAlpha 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.

3.1 Resolved, Medium: Use of Assert Instead of Require

Contract: Multiple

Explanation

Both the Basket and BasketEscrow contracts use `assert` in instances where `require` is recommended instead. The `assert` call is typically only used in code that will never fail, as if it is included in deployed code and a problem arises within `transfer` or `debundle` it could cause a failure. One key difference is that `assert` uses all remaining gas, and `require` returns any remaining gas, which is why `require` is preferred.

Resolution

The CoinAlpha team has replaced the `assert` statements with `require` statements.

3.2 Resolved, Medium: No Time Validation on Expiration

Contract: BasketEscrow

Explanation

Both `createBuyOrder` and `createSellOrder` allow a user to pass in an expiration time that has already passed. It cannot be filled and must be canceled, wasting both gas and on-chain storage.

Resolution

The CoinAlpha team has added a check for the expiration date via a `require` statement.

3.3 Resolved, Low: No Overflow Protection

Contract: BasketRegistry

Explanation

The `incrementBasketsMinted` function adds the amount of baskets being minted to the `totalMinted` amount without utilizing SafeMath or any other protections. Over time this amount could overflow and reset to 0.

Resolution

The CoinAlpha Team has implemented SafeMath's `add` function to protect against overflows.

3.4 Resolved, Low: No Overflow Protection

Contract: BasketRegistry

Explanation

The `incrementBasketsBurned` function adds the quantity of baskets being burned to the `totalBurned` amount without utilizing SafeMath or any other protections. Over time this amount could overflow and reset to 0.

Resolution

The CoinAlpha Team has implemented SafeMath's `add` function to protect against overflows.

---

4. Closing Statement

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

The CoinAlpha contracts create a system of baskets to bundle and unbundle tokens from all manner of cryptocurrencies. The CoinAlpha team has remedied issues in these contracts that, unchecked, could have allowed for the overflow of integers and wasted gas costs.

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 CoinAlpha 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: SafeMath

✓ Should skip operation on multiply by zero (62ms)

✓ Should revert on multiply overflow (61ms)

✓ Should allow regular multiple (65ms)

✓ Should revert on divide by zero

✓ Should allow regular division

✓ Should revert on subtraction overflow

✓ Should allow regular subtraction (42ms)

✓ Should revert on addition overflow

✓ Should allow regular addition (46ms)

Contract: ERC-20 Tests for TestToken

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

✓ Should allocate tokens per the minting function, and validate balances (52ms)

✓ Should `transfer` tokens from *0xd86543882b609b1791d39e77f0efc748dfff7dff* to *0x42adbad92ed3e86db13e4f6380223f36df9980ef* (170ms)

✓ Should not `transfer` negative token amounts (65ms)

✓ Should not `transfer` more tokens than you have (68ms)

✓ Should allow *0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3* to authorize *0x341106cb00828c87cd3ac0de55eda7255e04933f* to `transfer` 1000 tokens (92ms)

✓ Should allow *0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3* to zero out the *0x341106cb00828c87cd3ac0de55eda7255e04933f* authorization (171ms)

✓ Should allow *0x667632a620d245b062c0c83c9749c9bfadf84e3b* to authorize *0x53353ef6da4bbb18d242b53a17f7a976265878d5* for 1000 token spend, and *0x53353ef6da4bbb18d242b53a17f7a976265878d5* should be able to send these tokens to *0x341106cb00828c87cd3ac0de55eda7255e04933f* (413ms)

✓ Should not allow *0x53353ef6da4bbb18d242b53a17f7a976265878d5* to `transfer` negative tokens from *0x667632a620d245b062c0c83c9749c9bfadf84e3b* (140ms)

✓ Should not allow *0x53353ef6da4bbb18d242b53a17f7a976265878d5* to `transfer` tokens from *0x667632a620d245b062c0c83c9749c9bfadf84e3b* to 0x0 (134ms)

✓ Should not `transfer` tokens to *0x0* (55ms)

✓ Should not allow *0x53353ef6da4bbb18d242b53a17f7a976265878d5* to `transfer` more tokens than authorized from *0x667632a620d245b062c0c83c9749c9bfadf84e3b* (68ms)

✓ Should allow an approval to be set, then increased, and decreased (413ms)

Contract: Ownership Tests for TestToken

Deployment

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

Transfer

✓ Should not allow a non-owner to transfer ownership (49ms)

✓ Should not allow the owner to transfer to *0x0* (64ms)

✓ Should allow the owner to transfer ownership (133ms)

Contract: Pause Tests for TestToken

Deployment

✓ Should deploy in an un-paused state

Pause configuration

✓ Should be able to be paused (108ms)

✓ Should be able to be unpaused (184ms)

✓ Should not be able to be unpaused while unpaused (59ms)

✓ Should not be able to be paused while paused (145ms)

Contract: Destructible Tests for TestToken

Deployment

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

Transfer

✓ Should not allow a non-owner to `destroy` (41ms)

✓ Should not allow a non-owner to `destroyAndSend` (45ms)

✓ Should allow the owner to `destroy` (63ms)

✓ Should allow the owner to `destroyAndSend` (66ms)

Contract: Other Tests for TestToken

Deployment

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

Mint

✓ Should not allow a non-owner to `mint` (41ms)

✓ Should `mint` (120ms)

Faucet

✓ Should `faucet` tokens (108ms)

Fallback

✓ Should revert on fallback

Contract: Basket Tests

Deployment

✓ Should revert on invalid constructor (97ms)

depositAndBundle()

✓ Should `depositAndBundle` with no fee (630ms)

✓ Should fail to `depositAndBundle` with fee sent but no arranger fee (458ms)

✓ Should fail to `depositAndBundle` without enough ether for fee (585ms)

✓ Should `depositAndBundle` with enough ether for fee (726ms)

✓ Should fail to `depositAndBundle` with failing `transferFrom` (196ms)

debundleAndWithdraw()

✓ Should debundleAndWithdraw() (523ms)

✓ Should fail to debundleAndWithdraw() more than has been deposited (55ms)

burn()

✓ Should `burn` (311ms)

✓ Should fail to `burn` more than has been deposited (45ms)

withdraw()

✓ Should `withdraw` (566ms)

✓ Should fail to `withdraw` with failing transfer (468ms)

✓ Should fail to `withdraw` with no balance (42ms)

changeArrangerFeeRecipient()

✓ Should `changeArrangerFeeRecipient` (95ms)

✓ Should fail to `changeArrangerFeeRecipient` from non arranger

✓ Should fail to `changeArrangerFeeRecipient` to 0 (55ms)

changeArrangerFee()

✓ Should `changeArrangerFee` (77ms)

✓ Should fail to `changeArrangerFee` from non arranger (38ms)

() [fallback]

✓ Should fail to use fallback

Contract: BasketRegistry Tests

whitelistBasketFactory()

✓ Should fail to `whitelistBasketFactory` from nonadmin (46ms)

✓ Should `whitelistBasketFactory` (62ms)

checkBasketExists()

✓ Should `checkBasketExists`

getBasketDetails()

✓ Should `getBasketDetails` (260ms)

getBasketArranger()

✓ Should `getBasketArranger` (200ms)

incrementBasketsMinted()

✓ Should `incrementBasketsMinted` (248ms)

✓ Should fail to `incrementBasketsMinted` from non basket (215ms)

incrementBasketsBurned()

✓ Should `incrementBasketsBurned` (292ms)

registerBasket()

✓ Should fail to `registerBasket` from non BasketFactory (52ms)

✓ Should `registerBasket` (194ms)

✓ Should `registerBasket` multiple times from the same arranger (437ms)

() [fallback]

✓ Should fail to use fallback

Contract: BasketRegistry Tests

createBasket()

✓ Should `createBasket` (409ms)

✓ Should fail to `createBasket` without enough ether for fee (48ms)

changeProductionFeeRecipient()

✓ Should `changeProductionFeeRecipient` (81ms)

✓ Should fail to `changeProductionFeeRecipient` from non admin (40ms)

changeProductionFee()

✓ Should `changeProductionFee` (85ms)

✓ Should fail to `changeProductionFee` from non admin (40ms)

() [fallback]

✓ Should fail to use fallback (40ms)

Contract: BasketEscrow Tests

createBuyOrder()

✓ Should `createBuyOrder` (239ms)

✓ Should revert on duplicate `createBuyOrder` (335ms)

✓ Should fail to `createBuyOrder` for invalid basket (73ms)

createSellOrder()

✓ Should `createSellOrder` (460ms)

✓ Should fail to `createSellOrder` for unregistered basket (320ms)

✓ Should fail to `createSellOrder` with revert on `transferFrom` (120ms)

cancelBuyOrder()

✓ Should `cancelBuyOrder` without fee after expiration (183ms)

✓ Should `cancelBuyOrder` with fee before expiration (252ms)

cancelSellOrder()

✓ Should `cancelSellOrder` (258ms)

✓ Should revert `cancelSellOrder` that doesn't exist (76ms)

✓ Should revert `cancelSellOrder` that has already been filled (411ms)

fillBuyOrder()

✓ Should `fillBuyOrder` (499ms)

✓ Should revert on `fillBuyOrder` with failing `transfer` (188ms)

fillSellOrder()

✓ Should `fillSellOrder` (339ms)

✓ Should fail to `fillSellOrder` that doesn't exist (77ms)

✓ Should fail to `fillSellOrder` that has already been filled (386ms)

✓ Should fail to `fillSellOrder` that has expired (102ms)

getOrderDetails()

✓ Should `getOrderDetails` (72ms)

changeTransactionFee()

✓ Should `changeTransactionFee` (85ms)

✓ Should fail to `cancelBuyOrder` from non-Admin (40ms)

changeTransactionFeeRecipient()

✓ Should `changeTransactionFeeRecipient` (72ms)

✓ Should fail to `changeTransactionFeeRecipient` from non-Admin (39ms)

() [fallback]

✓ Should fail to use fallback (53ms)

6. Appendix B

All Contract Files Tested

Commit Hash: 935ea44aa1285add9f100d83b6cbfdec23ebd1ca

File	Fingerprint (SHA256)
Basket.sol	152586C5E8F17ED350643338E21AA2696F4C775A9176F7B130AA480F197D8EC0
BasketEscrow.sol	2038AE89C4E1D37746EC83086B8E996E95142519FA71BBEE3A6A4218A818B9A7
BasketFactory.sol	49A678E8ECA17E8F8A4B6BEF26FE1997C745ADF0B2C0E254796A34279AC2E4C5
BasketRegistry.sol	3E11F390BF888F30F7DCF25F927BDCDD8C4EEA1AEDFEC2649EE8C0C60C86544A
TestToken.sol	F4617959E5263BE3D7C4098E7A98F32A8DC0196D0118CF4F978BB7DFB7E31558
zeppelin/BasicToken.sol	29FF76339E274FF0A7E7383619A35062DF96919F7EC2E1BFDB0E679C55D837AA
zeppelin/Destructible.sol	5BB7197A51F5B08184A21B80592AABF0E17996B8664B04272BAB9AFEB55E2F85
zeppelin/ERC20.sol	53C6AF71322F1E0D7CB8B52D2F46005EF105E39A6E2151718DD7C690517BD12B
zeppelin/ERC20Basic.sol	A9CF1D9073A8A58CA6044A1720A93A69020EA80FAB3F5169192630590707D593
zeppelin/Ownable.sol	8205A18B0E715D0A4AB8151FEC31909B9D810CB136C572CE717FA51FA0AB4C85
zeppelin/Pausable.sol	A01796249805AF66B062F559D582AA90964210B2BD3274847F127EEC620AEBAB
zeppelin/SafeMath.sol	445A98BBC82A605285800C100BAD11C16F921542B59CD955A2198B2725B60541
zeppelin/StandardToken.sol	3A44CB5D15699E404CD108632B920CE340C7EB2165FD6E6523A5CC4A636EA010

7. Appendix C

Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
Basket.sol	100.00%	100.00%	100.00%	100.00%
BasketEscrow.sol	98.65%	90.00%	93.33%	100.00%
BasketFactory.sol	100.00%	100.00%	100.00%	100.00%
BasketRegistry.sol	100.00%	100.00%	68.75%	100.00%
TestToken.sol	100.00%	100.00%	100.00%	100.00%
zeppelin/BasicToken.sol	100.00%	100.00%	100.00%	100.00%
zeppelin/Destructible.sol	100.00%	100.00%	100.00%	100.00%
zeppelin/ERC20.sol	100.00%	100.00%	100.00%	100.00%
zeppelin/ERC20Basic.sol	100.00%	100.00%	100.00%	100.00%
zeppelin/Ownable.sol	100.00%	100.00%	100.00%	100.00%
zeppelin/Pausable.sol	100.00%	100.00%	100.00%	100.00%
zeppelin/SafeMath.sol	100.00%	100.00%	100.00%	100.00%
zeppelin/StandardToken.sol	100.00%	100.00%	100.00%	100.00%
All files	**99.60%**	**96.74%**	**91.89%**	**100.00%**