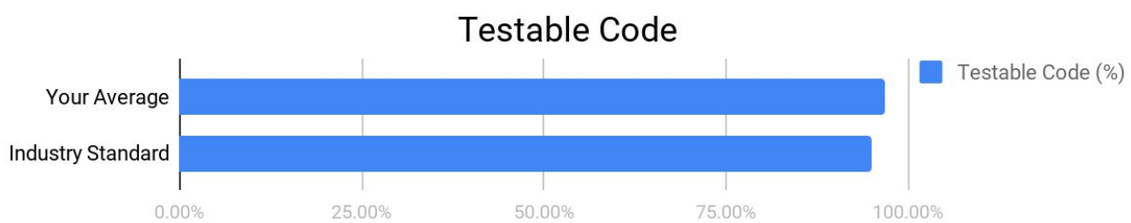
InfoCorp Token Contract Audit

by Hosho, March 2018

Executive Summary

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

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



We are pleased to report that the testable code is above industry average. (See Coverage Report)

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 InfoCorp 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 4

2.2 Coverage Report 5

2.3 Failing Tests 5

3. Complete Analysis **6**

3.1. Resolved, Medium: Airdrop Balance Transfer 6

Explanation 6

Resolution 6

3.2. Resolved, Medium: Unbounded Array Growth and Duplicate Entries 6

Explanation 6

Resolution 7

3.3. Resolved, Medium: Unclear Variable Value 7

Explanation 7

Resolution 7

3.4. Resolved, Medium: Variances in Whitelist Tests 7

Explanation 7

Resolution 7

3.5. Resolved, Low: Possible Race Condition 8

Explanation 8

Resolution 8

3.6. Resolved, Low: Non-Verified Order of Time Sensitive Array 8

Explanation 8

Resolution 8

3.7. Resolved, Low: No Protection Against 0x0 Value for Whitelist 8

Explanation 8

Resolution 8

3.8. Resolved, Low: Fallback Function Purchase Path 8

Explanation 9

Resolution 9

3.9. Resolved, Low: Gas Optimization 9

Explanation 9

Resolution 9

3.10. Resolved, Low: Potential Invalid End Time Value 9

Explanation 9

Resolution 9
3.11. Unresolved, Low: Vesting Start Function 9
Explanation 10
Resolution 10
3.12. Resolved, Informational: Incorrect Word Use 10
Explanation 10
Resolution 10
4. Closing Statement **11**
5. Test Suite Results **11**
6. All Contract Files Tested **17**
7. Individual File Coverage Report **18**

1. Auditing Strategy and Techniques Applied

The Hosho Team has performed multiple reviews of the smart contract code, the latest version as written and updated on March 8, 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 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.
- Is not affected by the latest vulnerabilities

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

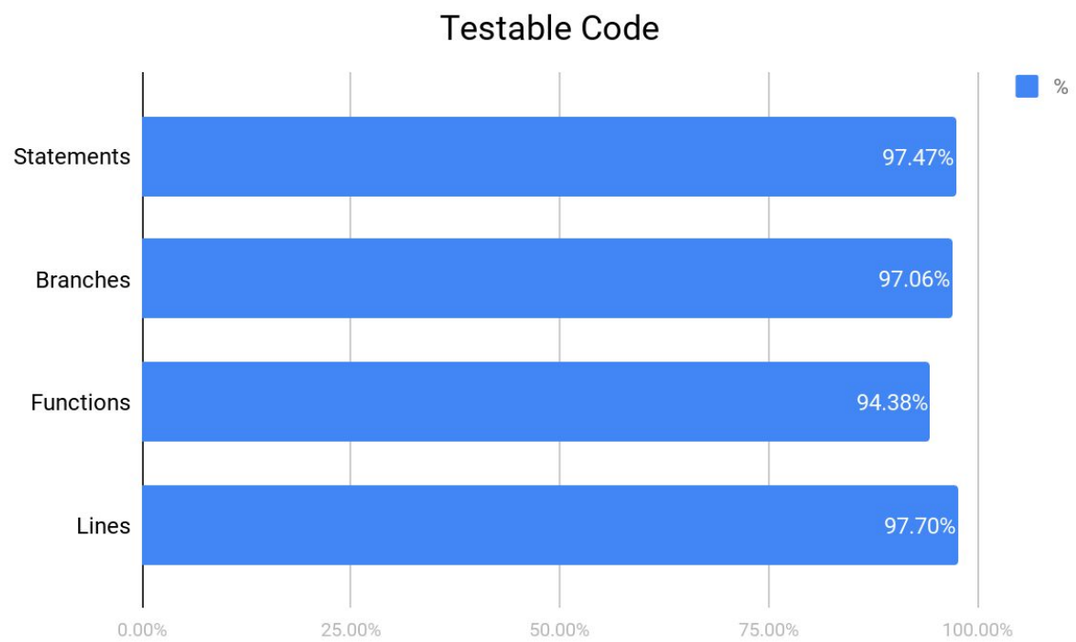
2.1. Summary

The SencToken and SencTokenSale are, respectively, an ERC-20 token and a crowdsale contract. The ERC-20 token is based on OpenZeppelin contracts, and the crowdsale contract has been written in-house. The token has been extended with a pausable system, allowing the token issuance to happen prior to any transfers. This pausable system has had the pause functionality disabled, ensuring that the token can not be halted in the future.

With the changes made by the SencToken team both for clarity and functionality, the contract is properly functional, and issues tokens at the expected rate. Utilizing a 800.00 USD per ETH value, or 800,000,000 USD/MEth, it was confirmed that the correct number of tokens (10,000 * 10^18) were issued.

2.2 Coverage Report

As part of our work assisting InfoCorp 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

1. Contract: Sale Tests for InfoCorp Administration. Should not allow the end time to be set to less than the current time. (See Issue 3.10)

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.

3.1. Resolved, Medium: Airdrop Balance Transfer

SencToken

Explanation

The following check: `while (i < dests.length && balances[bountyWallet] > values[i])`, forces the account balance of `bountyWallet` to always be greater than the overall amount airdropped. Due to this, the maximum airdrop will fail, for example, 50,000 tokens from a 50,000 token wallet.

Resolution

The comparison has been updated, allowing for the maximum airdrop to be completed.

3.2. Resolved, Medium: Unbounded Array Growth and Duplicate Entries

Whitelist

Explanation

Due to the design of `removeWhiteListed`, the address array becomes zeroed out at the location of the `addresses[addrIndex]`, but the entry itself is not removed. The `findWhiteListed` function indexes over this array, so if the array grows to contain excessive 0x0 address entries, this function will begin to run out of gas. While view functions require no gas to execute, they do have calculated gas costs and are required to remain within the gas limits of the network.

Resolution

The `findWhiteListed` view function has been removed, as well as the code path altered so that the array no longer needs to be iterated over, rather simply verified against known addresses, preventing this case.

3.3. Resolved, Medium: Unclear Variable Value

SencTokenSale

Explanation

Within SencTokenSale.sol, there is a section of code that supposedly handles the wei to USD conversions. However, due to the phrasing within the contract, it is not clear whether the values passed in, `ethusdRate`, represent the number of wei per USD or the current ETH/USD conversion rate. The code, on the other hand, appears to be calculating wei per USD cents, as entering a value under 10**18 will go to 0 due to the value being divided by 10**18. This creates the case where there is no usable value that we can put in to test these conversion.

Clarification is needed to properly execute and test.

Resolution

Clarification and code updates were provided by the InfoCorp Team, all tests are able to proceed and the conversion operates as intended.

3.4. Resolved, Medium: Variances in Whitelist Tests

SencTokenSale

Explanation

In some instances the allocations are checked against msg.sender, and in others, they're verified against the beneficiary. While this should not pose an issue due to the design of the fallback function, it is still a possibility that needs to be accounted for.

Resolution

All checks have been updated to verify against the beneficiary by the InfoCorp Team.

3.5. Resolved, Low: Possible Race Condition

Explanation

The ERC-20 implementation allows for the approval amount to be set again after it is initially set for an address. This creates a potential race condition where a new approved value can be set while there is still an existing approval amount.

Resolution

Acknowledged by the InfoCorp Team and external mitigations will be in place to prevent this case from occurring.

3.6. Resolved, Low: Non-Verified Order of Time Sensitive Array

SencTokenSale

Explanation

There is no validation for the order of the array in the function `BatchStartTimes`. This allows the case to be created where the first index is after the second index such as: [1000, 2000, 1500], making it possible that a batch may never become active because the end time verification is based off the last entry during the initialization of the contract.

Resolution

Order validation on the array is now included in the contract.

3.7. Resolved, Low: No Protection Against 0x0 Value for Whitelist

SencTokenSale

Explanation

The system defines an interface in the initial call, however, since it does not call any value from the remote contract, it impossible for the contract to determine if the remote contract is in fact a whitelist or not.

Resolution

Protections against the improper values for the whitelist call have been added to the contract.

3.8. Resolved, Low: Fallback Function Purchase Path

SencTokenSale

Explanation

Due to the mechanics of `.send()` versus `.call()` used with a fallback function, if a contract such as a multi-sig wallet uses `.send()`, it will be unable to purchase tokens due to the low amount of gas transferred.

Resolution

Acknowledged by the InfoCorp Team.

3.9. Resolved, Low: Gas Optimization

SencTokenSale

Explanation

The expense in calling a remote contract for data, as well as loading data from various parts of the chain is excessively high. A more gas efficient path is to check if the msg.value is over the MIN_CONTRIBUTION value then validate before any requires are called.

Resolution

The InfoCorp Team implemented gas optimization techniques, including the Hosho recommendation.

3.10. Resolved, Low: Potential Invalid End Time Value

SencTokenSale

Explanation

There are no protections against the end time being set to a value that is less than the current time. This allows the instant halting of the contract which blocks all sales.

Resolution

As the sales contract was already deployed and unable to be altered, the InfoCorp Team is aware of this issue and will take steps to mitigate externally.

3.11. Unresolved, Low: Vesting Start Function

SencVesting

Explanation

Due to the design of the system, if deposits are made into the vesting contract that are not followed by the corresponding `addEntry` call, the voting start function may be difficult to execute.

Resolution

The InfoCorp Team is aware of this issue and will prevent this from occurring.

3.12. Resolved, Informational: Incorrect Word Use

Operatable

Explanation

Operatable should be Operable.

Resolution

Acknowledged by the InfoCorp Team and unaltered.

---

4. Closing Statement

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

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

5. Test Suite Results

Uncovered Code:

SencTokenSale - Contains a single uncovered branch. With the validation of the recipient of the token not being 0x0, as msg.sender is used for sending in the recipient to the internal function, there is no way to test this branch. However, it is self-protective and thus not a testing concern.

Contract: ERC-20 Tests for SencToken

✓ Should deploy a token with the proper configuration (135ms)

✓ Should allocate tokens per the minting function, and validate balances (607ms)

✓ Should transfer tokens from 0xd86543882b609b1791d39e77f0efc748dfff7dff to 0x42adbad92ed3e86db13e4f6380223f36df9980ef (120ms)

✓ Should not transfer negative token amounts (45ms)

✓ Should not transfer more tokens than you have (42ms)

✓ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens (72ms)

✓ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization (68ms)

✓ Should allow 0x667632a620d245b062c0c83c9749c9bfadf84e3b to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and 0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to 0x341106cb00828c87cd3ac0de55eda7255e04933f (247ms)

✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer negative tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (39ms)

✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b to 0x0 (38ms)

✓ Should not transfer tokens to 0x0

✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer more tokens than authorized from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (49ms)

✓ Should allow an approval to be set, then increased, and decreased (303ms)

✓ Should allow the owner to set the operators, and validate the operator's state (143ms)

✓ Should not permit minting once Minting is complete

✓ Should not permit an airdrop where destinations isn't equal to values

✓ Should not allow airdrop if the approve function hasn't been completed (66ms)

✓ Should correctly handle a failed airdrop due to the `transferFrom` failing (110ms)

✓ Should airdrop tokens properly when approval is given, and the transfer does not exceed the approval (215ms)

Contract: Token Reclaim Tests for SencToken

Token reclaim

✓ Should allow the reclaim of tokens from the tested contract (94ms)

✓ Should safely handle a failed token transfer (95ms)

✓ Should attempt to return ETH (75ms)

✓ Should only allow an operator to execute an emergency withdrawal/escape

Contract: Ownership Tests for SencToken

Deployment

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

Transfer

✓ Should not allow a non-owner to transfer ownership

✓ Should not allow the owner to transfer to 0x0

✓ Should allow the owner to transfer ownership (85ms)

Contract: Pause Tests for SencToken

Deployment

✓ Should deploy in an paused state

Pause configuration

✓ Should be able to be unpaused (151ms)

✓ Should not be able to be unpaused while unpaused (97ms)

✓ Should not be able to be paused

Contract: Sale Tests for InfoCorp

Deployment

✓ Should not deploy with 0-length batch times (228ms)

✓ Should not deploy with batch times out of order (170ms)

✓ Should not deploy with an end time before the last batch times (164ms)

✓ Should require that the `ethUSD` rate is > 0 (175ms)

✓ Should not allow wallet addresses of 0 for any initial minting wallets (953ms)

✓ Should not allow the Whitelist address to be 0 (167ms)

Administration

✓ Should return the number of start times in the batch

✓ Should allow the batch start time to be set (61ms)

✓ Should allow the end time to be set (67ms)

X Should not allow the end time to be set to less than the current time

Events emitted during test:

Mint(to: <indexed>, amount: 1e+26)

Transfer(from: <indexed>, to: <indexed>, value: 1e+26)

EarlySupporters(purchaser: 0x14bcb572662c014b7ec94c2d2058d937a3c088d9, amount: 1e+26)

Mint(to: <indexed>, amount: 1e+26)

Transfer(from: <indexed>, to: <indexed>, value: 1e+26)

TokenPresale(purchaser: <indexed>, amount: 1e+26)

Mint(to: <indexed>, amount: 1.5e+26)

Transfer(from: <indexed>, to: <indexed>, value: 1.5e+26)

TokenTreasury(purchaser: 0x524646ee200d0f2665b9e8c6786fd9aa85ce4f50, amount: 1.5e+26)

Mint(to: <indexed>, amount: 5e+25)

Transfer(from: <indexed>, to: <indexed>, value: 5e+25)

TokenFoundingTeam(purchaser: 0x9ab4ec6e01306405bb00b5c1d2cb473aa30c7cab, amount: 5e+25)

✓ Should not allow the MEth/USD rate to be changed after the first `batchStartTime` is hit (245ms)

✓ Should allow the MEth/USD rate to be changed before the first `batchStartTime` is hit (102ms)

✓ Should return back the ended state (259ms)

✓ Should only allow the contract to be finalized once (518ms)

✓ Should return the correct batch active at any given time (1261ms)

✓ Should return back the eth raised

✓ Should return back the usd raised (40ms)

✓ Should return back the number of sold tokens

Purchasing

✓ Should not allow purchase if the buyer isn't whitelisted (42ms)

✓ Should not allow purchase if the buyer is whitelisted, but isn't in the right period (55ms)

✓ Should not allow purchase if the contract is ended (260ms)

✓ Should not allow purchase if the amount is below 120 finney (72ms)

✓ Should not allow purchases over the maximum whitelisted contribution amount (84ms)

✓ Should purchase the correct number of tokens (270ms)

✓ Should allow tokens to be purchased in multiple batches (399ms)

Contract: Whitelist Tests for InfoCorp

✓ Should require that the address and batch length are the same (120ms)

✓ Should require that the address and `weiAllocation` length are the same (45ms)

✓ Should only allow an address to be whitelisted once during an import (162ms)

✓ Should allow a user to be re-added to the whitelist, after being removed (279ms)

✓ Should require that resetting allocations for a batch of addresses has equal argument lengths

✓ Should require that a user is whitelisted before de-whitelisting them

✓ Should require that resetting batch numbers for a batch of addresses has equal argument lengths

✓ Should allow the allocation on an address to be edited (138ms)

✓ Should allow the batch number on an address to be edited (232ms)

Contract: SencVesting

✓ Should require contract be deployed with a SencToken instance (72ms)

✓ Should allow the reclaim of tokens from the tested contract (99ms)

✓ Should allow the reclaim of tokens from other ERC20 token after vesting has started (284ms)

✓ Should require vesting to not be started to the reclaim of tokens (325ms)

✓ Should require vesting to not be started to add vesting entry (227ms)

✓ Should require vesting entry to be a valid beneficiary address (48ms)

✓ Should require vesting entry token amount to be greater than zero (45ms)

✓ Should require vesting entry period to be greater than zero (57ms)

✓ Should require vesting entry to have no existing entry for the beneficiary (117ms)

✓ Should require vesting to not already be started for starting (220ms)

✓ Should require a total token balance to start vesting (40ms)

✓ Should require token balance to match total tokens to start vesting (360ms)

✓ Should allow vesting with a 24 week period (164ms)

✓ Should allow vesting with a specified period in seconds (148ms)

✓ Should store correct number of vested tokens (153ms)

✓ Should report zero withdrawable amount if vesting has not been started (198ms)

✓ Should require vesting entry for any withdrawal

✓ Should require vesting to have started before allowing withdraw (196ms)

✓ Should report fully vested once total vesting period has passed (453ms)

✓ Should report the correct vested amount after vesting as started (542ms)

✓ Should allow correct withdraw amount for a given vesting period (612ms)

✓ Should require token transfer to succeed in a withdrawal before invoking Withdrawn event (523ms)

✓ Should allow owner to withdrawal on behalf of another account (604ms)

✓ Should report the correct vested amount for beneficiary with no entry (464ms)

6. All Contract Files Tested

File	Fingerprint (SHA256)
contracts/Operatable.sol	91330412e1c007ab2a14f89977df0b00668ec358e561e14c4a34a784f8eab51b
contracts/OperatableBasic.sol	d997773c39c4ee21b122ce57a4127d35b00597392a67df9771ac011260a89935
contracts/Salvageable.sol	0f48e9ef2fd0a58a7be49186d9c119e66516c93ac01f8edbd1c563f31517f132
contracts/SencToken.sol	4554bea320a69bf9c06c98d92d71b9468c3e6fe395e8a5773b95a681475f7089
contracts/SencTokenConfig.sol	e89f53ba479436f8f3d72295e978332eb1fd7a43111691ee2d69535d83ae1b09
contracts/SencTokenSale.sol	3bcdc00e5aadaf8786130bf261b96c5ce36cad0ccfe7b239cfd058af0d0dec2f
contracts/SencTokenSaleConfig.sol	1d48b29fdb1f1b0509979af52c43b19dd844d3cfd272d6a6b6d61434f6ce4928
contracts/SencVesting.sol	1dd55f7de46f81f6699690104541e1e898dfcc36fdae39d35aafb590ab39a54a
contracts/WhiteListed.sol	9633c97a437610e34ffa35cc8c7a2032611451887a8278a2e24c968a0df3925a
contracts/WhiteListedBasic.sol	d7cd8e08de3c7e012c178971e6b74ee5408838379a39425f158915b413db4121
contracts/zeppelin-solidity/contracts/lifecycle/Pausable.sol	78bf21e029fc3f1c38151915db9ccce2f0553bfeae9b6685fde1c297091cdb6f
contracts/zeppelin-solidity/contracts/math/Math.sol	fc23802a0629f896bd3553a0df596de799e8f82aebd644101369433a9229dc27
contracts/zeppelin-solidity/contracts/math/SafeMath.sol	e434336813af116101008bcfaed8cc02fa051c9c2b612a477b6bfa0765fa17f6
contracts/zeppelin-solidity/contracts/ownership/Ownable.sol	35dcf237365077adb1dd8d1da9e05f2b4f8e9d7b49311fc8a09b28d4ce191579
contracts/zeppelin-solidity/contracts/token/ERC20/BasicToken.sol	2662ea846c0c6ca2bd32dfdb97ab20ae4108e7abade53962f146f634e9f4eba7
contracts/zeppelin-solidity/contracts/token/ERC20/ERC20.sol	6b75acd05c29968b057ec1facf659c064dbe0a79ac01444530629f01ef3a3abf
contracts/zeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol	86c0a5fc6cb564ae77140da57a8ff9a22f46404240e69a6782ff741e286d373a
contracts/zeppelin-solidity/contracts/token/E	2686975792c8dc8f507dc143aa0abb7f49eb837c8bde51017216dae3fa38dafd

RC20/PausableToken.sol	
contracts/zeppelin-solidity/contracts/token/ERC20/StandardToken.sol	77e45da1164753f886d7395987b46deb036eca32c2e7322ef7a2764a08f7c5da

7. Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
contracts/Operatable.sol	100.00%	100.00%	100.00%	100.00%
contracts/OperatableBasic.sol	100.00%	100.00%	100.00%	100.00%
contracts/Salvageable.sol	100.00%	100.00%	100.00%	100.00%
contracts/SencToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/SencTokenConfig.sol	100.00%	100.00%	100.00%	100.00%
contracts/SencTokenSale.sol	100.00%	97.83%	100.00%	100.00%
contracts/SencTokenSaleConfig.sol	100.00%	100.00%	100.00%	100.00%
contracts/SencVesting.sol	100.00%	100.00%	100.00%	100.00%
contracts/WhiteListed.sol	100.00%	100.00%	100.00%	100.00%
contracts/WhiteListedBasic.sol	100.00%	100.00%	100.00%	100.00%
contracts/zeppelin-solidity/contracts/lifecycle/Pausable.sol	66.67%	100.00%	75.00%	75.00%
contracts/zeppelin-solidity/contracts/math/Math.sol	0.00%	100.00%	0.00%	0.00%
contracts/zeppelin-solidity/contracts/math/SafeMath.sol	100.00%	62.50%	100.00%	100.00%
contracts/zeppelin-solidity/contracts/ownership/Ownable.sol	100.00%	100.00%	100.00%	100.00%
contracts/zeppelin-solidity/contracts/token/ERC20/BasicToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/zeppelin-solidity/contracts/token/ERC20/ERC20.sol	100.00%	100.00%	100.00%	100.00%
contracts/zeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol	100.00%	100.00%	100.00%	100.00%
contracts/zeppelin-solidity/contracts/token/ERC20/PausableToken.sol	100.00%	100.00%	100.00%	100.00%
contracts/zeppelin-solidity/contracts/token/ERC20/StandardToken.sol	100.00%	100.00%	100.00%	100.00%
All files	**97.47%**	**97.06%**	**94.38%**	**97.70%**