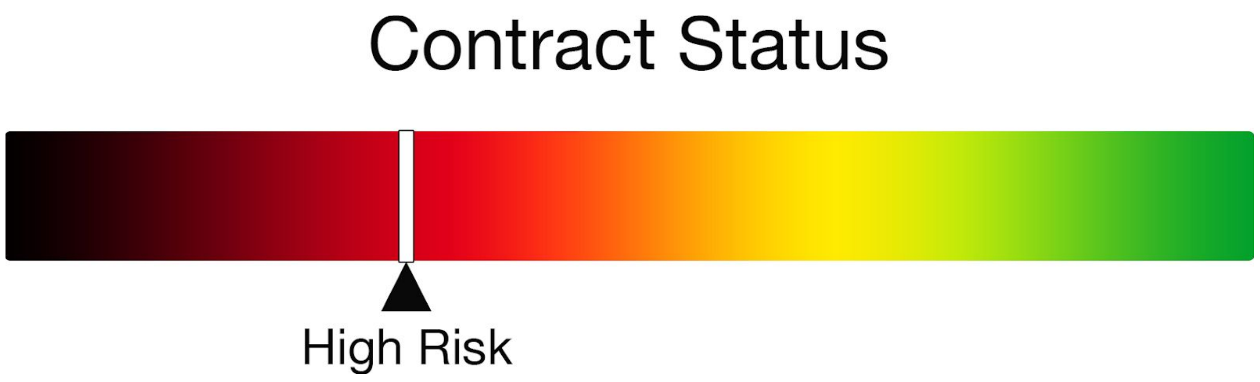
Codex Protocol Registry Contract Audit

Prepared by Hosho
July 10th, 2018

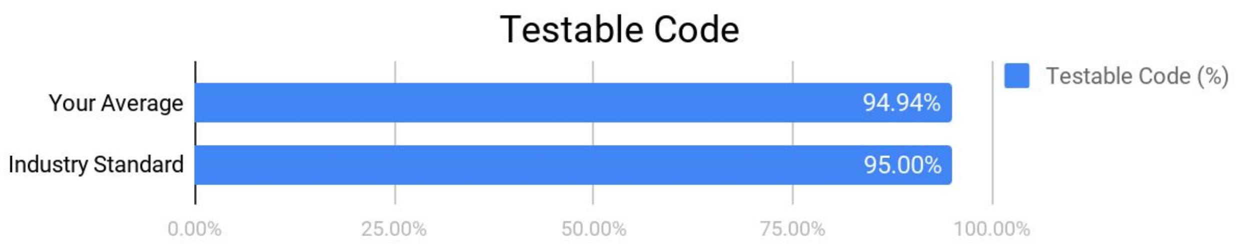
Report Version: 2.0

Executive Summary

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



There is one high level issue within these contracts, as well as a low issue introduced in the remediated code. (See Complete Analysis)



Testable code is 94.94% which is on par 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 Codex Protocol 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 9th, 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;
- 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 Codex Protocol's contracts. 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

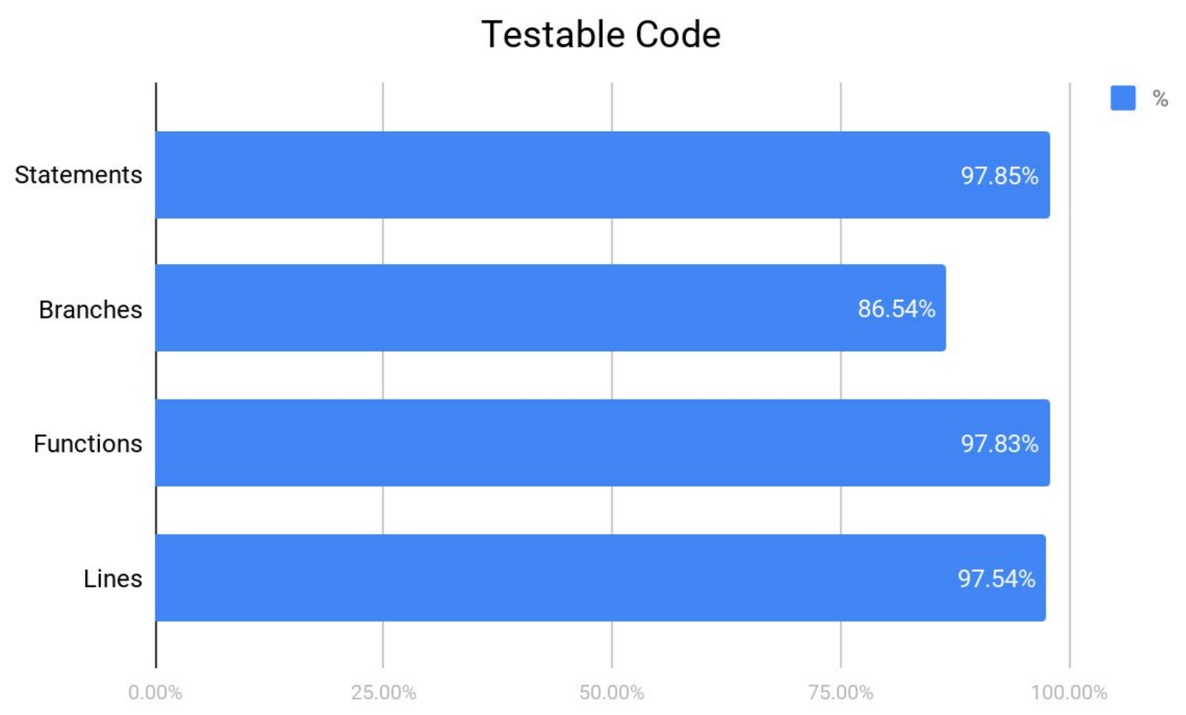
2.1. Summary

The Codex Registry contracts are a series of token contracts based around an ERC-721 token, which is upgradable via a proxy contract. The ERC-20 tokens, contained in a separate contract from the Registry, are able to be vested into an ERC-900 staking interest contract, which grants the token owner continually reduced transfer fees, until there are no fees at all. The staking interest contract also creates the ability to accrue annual interest on the ERC-20 tokens, after enough tokens have been vested in the contract.

Aside from the high severity issue discussed with the Codex Protocol team, there is a low issue regarding wasted gas that was introduced in the remediated code, detailed in Issue 3.6.

2.2 Coverage Report

As part of our work assisting Codex Protocol 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-721 Tests for ERC-721 TokenMock using `transferFrom` should not be able to transfer token to owner (See Issue 3.8 Allows `transferFrom` to Owner)
2. Contract: ERC-721 Tests for ERC-721 BasicTokenMock using `transferFrom` should not be able to transfer token to owner (See Issue 3.8 Allows `transferFrom` to Owner)

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: Repeat Ownership Changes

Contract: DelayedOwnable

Explanation

The `initializeOwnable` function should only be called once. This function does not set the `isInitialized` state after an owner has been assigned, allowing ownership to be changed repeatedly.

Resolution

The Codex Protocol Team has added the line `` `isInitialized = true` `` after calling the `initializeOwnable` function. As there is no other place in the code allowing this state change, by including this code, ownership will no longer be able to be changed.

3.2 Unresolved, High: Potential Ability to Lose Ownership

Contracts: CodexStakeContainer and CodexRecord

Explanation

These contracts use `DelayedOwnable` to manage its ownership, which means there is no explicit initial setup for ownership. As this is normally handled by `initializeOwnable` in the `DelayedOwnable` contract, the contract does not initialize with an owner. If the owner forgets to initialize ownership, or is raced for ownership by a third party, anyone could call `initializeOwnable` to become the owner of this contract.

Update

The Hosho auditing team has lowered this issue from a Critical severity down to a High severity after speaking with the Codex Protocol team. They have verified that extra precautions against this case will be implemented and the contract will not be deployed in a state that allows ownership changes.

3.3 Resolved, High: Incorrect Type Comparison

Contract: CodexRecordMetadata

Explanation

Within the `modifyMetadataHashes` function there is an integer comparison to verify that a *bytes32* is empty. If this remains not casted to the proper type, it will cause hard reverts and errors within the EVM.

Resolution

The Codex Protocol team has added new functionality to this contract verifying the entire array is empty, rather than the first element as was previously implemented.

Technical Note

There is a common misconception that the *bytes32* data-type is functionally a fixed-length string, but this is not the case. While it is true that the first character of a string can never be null, or else a string is null, a raw hex value is able to be passed into a *bytes32* variable which would cause unintended results. For example, if the values *0x0001* or *0x0074* were passed in, `bytes32[0]` would be equal to *00* even though both of these values are non-empty bytes objects.

3.4 Resolved, High: Value Mismatch

Contract: RC900BasicStakeContainer

Explanation

The `annualizedInterestRate` does not match the code comments. If the `interestRate` is 10 after 1 year, the perceived stake would be 1/1e17 more valuable, instead of 10% more valuable as noted in the contract.

Resolution

The Codex Protocol team has update their comments to match the functionality of the related code.

3.5 Resolved, High: Gas Limit

Contract: ERC900BasicStakeContainer

Explanation

Due to the loops built into the `updatePerceivedStakeAmounts` function, it is possible that this function can exceed the gas limit for the block, causing it to be unable to finalize.

Resolution

The Codex Protocol team has limited the elements being iterated over, in the loop found in the `updatePerceivedStakeAmounts` function, to strictly the stakes that have not been updated in the current year, reducing the total amount of gas used and greatly reducing the possibility of exceeding the gas limit.

3.6 Unresolved, Low: Wasted Gas

Contract: CodexRecordMetadata

Explanation

When the `getByTokenId` function is called using an invalid token ID, there is no revert to handle this case, which causes unnecessary gas to be burned.

3.7 Resolved, Informational: Unnecessary Import

Contract: CodexStakeContainer

Explanation

The CodexStakeContainer contract inherits the Pausable contract, but it does not utilize any pausable functions or modifiers.

Resolution

The Codex Protocol team have replaced the CodexStakeContainer with CodexStateContract which has removed the unnecessary import.

3.8 Resolved, Informational: Allows transferFrom to Owner

Contract: ERC721BasicToken

Explanation

The `transferFrom` function allows for tokens to be transferred from the owner of the token, back to themselves.

Resolution

The Codex Protocol team has indicated that they are aware of this functionality and have decided to leave it in for potential future use cases.

3.9 Resolved, Informational: Lack of Ownership Modifiers

Contract: CodexRecordProxy

Explanation

Delegate call utilization should be protected behind ownership modifiers, as it allows remote contracts to execute code that can modify the local storage state of the contract. The severity level of this risk can potentially be reduced if implementation details are provided by the Codex Protocol team about the expected protections around this call.

Update

Through discussions with the Codex Protocol team, they will be placing external protections around this call and are mitigating all of the risks that are inherent in this type of call with a full understanding of the operation of this functionality.

---

4. Closing Statement

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

While there is still a high severity issue that remains within the system, the team is aware of this and will be mitigating any associated risks externally. Overall, these contracts operate as intended and have followed sound programming practices. Any concerns that were raised during the auditing process were responded to and taken care of promptly. We look forward to working with the Codex Protocol team in the future.

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 Codex Protocol 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

<u>Contract: CodexStakeContract Tests</u>

When properly deployed

√ Owner should have the `initialBalance`

√ Should show not `supportsHistory`

√ Should use token to check `stakingToken` address

PersonalStake

√ Should `getPersonalStakeUnlockedTimestamps` (61ms)

√ Should `getPersonalStakeActualAmounts` (58ms)

√ Should `getPersonalStakeForAddresses` (56ms)

Stake and stakeFor function

√ Should `transfer` token when staked

√ Should `transfer` token by using stakeFor (248ms)

√ Should not be able to stake when account doesn't have `staketoken` (164ms)

Unstake

√ Should unstake token (113ms)

√ Should not be able to unstake token when the current stake the current stake is locked (59ms)

√ Should not be able to unstake token when the unstake amount does not match the current stake (39ms)

√ Should not be able to unstake when unable to withdraw stake (155ms)

SetDefaultLockInDuration function

√ Should be able to `resetDefaultLockInDuration` (51ms)

<u>Contract: metahashMock</u>

deploy

√ Should deploy a token with the proper configuration (63ms)

√ Should use `getTokenById`

√ Should not use `tokenOfOwnerByIndex` by using the wrong index

√ Should not use `tokenOfOwnerByIndex` by wrong account

√ Should not use `tokenByIndex` by using the wrong index

new change cause vulnerability

√ Should not change the name and symbol (42ms)

1) Should not be able to use `getTokenById` get invalid Token

mint

√ Should able to mint a no `_providerId` and `_providerMetadataId` token (146ms)

transferFrom

√ Should able to `transfer` token by owner (203ms)

safeTransferFrom

√ Should able to `safetranfer` token by owner (217ms)

√ Should be able to `safetranfer` token with data (196ms)

modifyMetadataHashes

√ Should able to use `modifyMetadataHashes` to modify token when the first digit of `_newNameHash` and `_newFileHash` is 0x00 (335ms)

√ Should not modify token by calling `modifyMetadataHashes` when the `_newNameHash` and `_newFileHash` is *0x0* (603ms)

tokenURI

√ Should `setTokenURIPrefix` (116ms)

√ Should get "" when `calltokenURI` if do not setup `tokenPrefix`

√ Should able to use 0 as `tokenId` to call `tokenURI` (90ms)

fix 1.1

vulnerability

√ Should not call `initializeOwnable` again after setup owner (86ms)

√ Should be able to `transferOwnership` by owner (106ms)

√ Should not be able to `transferOwnership` to 0x0 (86ms)

√ Should not be able to `transferOwnership` if the account is not owner (83ms)

transfer Fee

√ Should `setFee` by owner (69ms)

√ Should `setStakeContainer` by owner (43ms)

test transfer with fee

√ Should charge fee when `transfer` (351ms)

√ Should not `transfer` token without paying transfer fee (133ms)

√ Should not change `transferFee` when transfer fee change to 0 (349ms)

Contract: Pause Tests for DelayedPausableToken

Deployment

√ Should deploy in an un-paused state

Pause configuration

√ Should be able to be paused (56ms)

√ Should be able to be unpaused (116ms)

√ Should not be able to be unpaused while unpaused

√ Should not be able to be paused while paused (75ms)

Contract: Proxy Tests for ERC721TokenMock

Deploy

√ Should have the proper eth balance

Proxy functions

√ Should call name

√ Should call name

upgradeTo

√ Should able to `upgrade` to a higher version (124ms)

√ Should not be able to `upgrade` to same version (38ms)

√ Should not be able to `upgrade` to address Zero (52ms)

Proxy storage

√ Should get the last version storage

√ Should able to run the unique function of v2 (111ms)

Fallback

√ Should trigger the fallback when the function is not below to CodexRecordProxy.sol (54ms)

√ Should trigger the fallback revert when `delegagtecall` has problem

√ Should trigger revert when directly send eth to contract

√ Should also not trigger fallback by use `name` and `symbol` (48ms)

Contract: CodexRecord

√ Should reclaim tokens from a ERC20 contract (60ms)

√ Should revert if a non-owner attempts to reclaim tokens from an ERC-20 contract

Contract: ERC-721 Tests for ERC721TokenMock

Deploy

√ Should deploy a token with the proper configuration (88ms)

√ Should not use `tokenOfOwnerByIndex` by using the wrong index

√ Should not use `tokenOfOwnerByIndex` by wrong account

√ Should not use `tokenByIndex` by using the wrong index

Check interface Id

√ Contract should have correct ERC165 Id

√ Contract should have correct INTERFACE_ERC721 Id

√ Contract should have correct INTERFACE_ERC721_ENUMERABLE id

√ Contract should have correct INTERFACE_ERC721_METADATA id

SupportsInterface

√ Should return support ERC165

√ Should return support ERC721

√ Should return support ERC165_ENUMERABLE

√ Should return support ERC721_MetaEData

tokenUri

√ Should set URI for token (99ms)

√ Should not be able to set URI for wrong token (43ms)

√ Should not able to get token URI for not existing token

Using transferFrom

√ Should able to `transfer` token by owner (181ms)

√ Should able to `transfer` token from approved account (197ms)

√ Should not be able to `transfer` token from *ZERO_ADDRESS* (52ms)

√ Should not be able to `transfer` token to *ZERO_ADDRESS* (56ms)

√ Should not be able to `transfer` token after the token doesn't belong to the sender (127ms)

<u>Contract: ERC-721 Tests for ERC721BasicTokenMock</u>

SupportsInterface

√ Should return support ERC165

√ Should return support ERC721

Mint

√ Should not be able to mint token to *ZERO_ADDRESS*

√ Should not be able to mint token which already been minted

With proper distribution

√ Should give *0x1bbb1269032bfd0b0fe0851235fc798af6bd3c9b* proper token

√ Should give *0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9* proper token

√ Should not able to find accounts balance of *ZERO_ADDRESS*

Using ownerOf

√ Should find out the owner by proper `tokenId`

√ Should not find out the owner by wrong `tokenId`

Approve

√ Should able to approve to `transfer` the token to another account by owner (83ms)

√ Should not approve to *ZeroAdress* (78ms)

√ Should not be able to approve to `transfer` the token to itself (43ms)

√ Should not be able to approve to `transfer` the token by the accounts do not have authorization (44ms)

Using setApprovalForAll

√ Should able to `approve` operator by owner (61ms)

√ Should not be able to `approve` owner it self

√ Should able to disapprove the operator by owner (112ms)

Using transferFrom

√ Should able to `transfer` token by owner (116ms)

√ Should able to `transfer` token from approved account (133ms)

√ Should not be able to `transfer` token from *ZERO_ADDRESS* (47ms)

√ Should not be able to `transfer` token to *ZERO_ADDRESS* (51ms)

2) Should not be able to `transfer` token to owner

√ Should not be able to `tranfer` token after the token donot belong to the sender (94ms)

Using safeTransferFrom

To safetransfer token to accounts

√ Should able to `safetransfer` token by owner (134ms)

√ Should be able to `safetransfer` token with data (127ms)

to safetransfer token to contract address

√ Should be able to `safetransfer` token to contract address (135ms)

√ Should be able to `safetransfer` token to contract address with data (126ms)

√ Should not be able to `safetransfer` token to invalid contract address (149ms)

√ Should not be able to `safetransfer` token to contract address when receiver contract revert (175ms)

<u>Contract: SafeMath</u>

√ Should skip operation on multiply by zero

√ Should revert on multiply overflow

√ Should allow regular multiple

√ Should revert on divide by zero

√ Should allow regular division

√ Should revert on subtraction overflow

√ Should allow regular subtraction

√ Should revert on addition overflow

√ Should allow regular addition

<u>Contract: ERC900CompoundingStakeContract test</u>

√ Should know the annual perceived stake is `annualizedInterestRate`/1e18 (225ms)

√ Should allow users to unstake tokens (142ms)

<u>Contract: ProxyOwnable Tests for ProxyOwnable</u>

Deployment

√ Should deploy in an un-paused state

TransferProxyOwnership

√ Should transfer ownership (43ms)

√ Should not allow transfer ownership to address 0

√ Should not allow no owner accounts to transfer ownership

6. Appendix B

All Contract Files Tested

Commit Hash: 9fd0093ef170e5003c31cadcf746dce276e2f632

File	**Fingerprint (SHA256)**
CodexRecord.sol	8871ba114a484ef2283e54c665ed1816269ad5aad12a62ebb359af3f868288ba
CodexRecordAccess.sol	d174b34150d9ccacac3eaaee65b6986cda79c0e43c55651b303bb3857e0787b3
CodexRecordCore.sol	b372cb4e5a7cced3237ea13c49c7fe101292c6e9ed07e311f18320ecd1cedff0
CodexRecordFees.sol	0695715ae71fb350e2e136c8868434c59351a93d54e0e327b09b9230028ec528
CodexRecordMetadata.sol	a0afe7241595a71e43b27e8e43eab6bdfdfccc37bf65d5b60a988ace40b46267
CodexRecordProxy.sol	621f04ac129526a66bf9d60810fd596ab0e1ec83f1cd6e7685e2c2631eaf0feb
CodexStakeContract.sol	99830f85cdf577bd1e3e14b9058cb116d06794b11925244b35a4cde43a201d18
CodexStakeContractInterface.sol	be84035e3e086d33db49625ed3e9b070b66c66fd1b24057221e91cf150cbd717
ERC20/ERC20.sol	208887eaf452703c4aa08e92d35e6842524b817bfed07dce6b10dde28e24a0fc
ERC20/ERC20Basic.sol	00061d6873af6c4dd1298035aefed24eefd0770a3ba48c73929de7306a70d9f7
ERC165/ERC165.sol	05b7f89f387308b54c052968684faa1878e71d70d5b55aa8fb7127beaa612a96
ERC721/ERC721.sol	9f2e7aed0107cee06619367cb132707c922f7fd848ebac8f36e63d141d7eccb4
ERC721/ERC721Basic.sol	56bb5c9f42a7646b426048725bca1240da32485d153971fbb668c6afc5351458
ERC721/ERC721BasicToken.sol	9ffc0e3466a7b071d4727503672db0d385beaaa70f24f990229131e4cc2faf65
ERC721/ERC721Receiver.sol	3f8e76044b2a714026e37d11e74db0b30c3e748256a0f997e7074bbb56a71cd4
ERC721/ERC721Token.sol	e92584cf0af09fc77c8673e3098c12f6332ab4e733d9bc66c6ac2e361e1d9650
ERC900/ERC900.sol	a47944d56942ece2644f88437eef71c5c582044db7b64fc8842670cb93f41560
ERC900/ERC900BasicStakeContract.sol	70e307be54505b2e13b00d4ef507ba02cd341ebcdaaa0c59296a9a7f5ab3fa6c
ERC900/ERC900CompoundingStakeContract.sol	107f1fe6e822925e2dbae40843af242fe9648a04f343c041fae458558cac337b

ERC900/ERC900CreditsStakeContract.sol	4bf5383a7537c59638d4eb83935f40daf071e8510d87d15c34e874f0c0ee009c
library/AddressUtils.sol	6908e5adb625d0f4b144080cb1a7020428076f14e43865c75166cbe06fbc5bf3
library/Debuggable.sol	d24aad521ad0c13a0065597c9fdc7b38d863ded603b8cd348f6f3acbc8ba5021
library/DelayedOwnable.sol	01ed8ca931476cf8f7bea3a3de2b5a67588e905d61473be5023088cbca9dc957
library/DelayedPausable.sol	7e7fcf6503e3c5acd0dbb18d94416b6330e6cb53df66469027b5d52a494b6b98
library/Ownable.sol	2a98a66a54a81df01855abe695645bc03ac347d300de15c7f4d2d0de3773b38d
library/Pausable.sol	55a28478ee48c63c74cd2db488ebe7dccc5dec635b6bb7cf13fdcd076e192513
library/ProxyOwnable.sol	04236193bab87d031bc6597c8bce85e0a6ef9d9a73d04a32fbeb3c0685c942cf
library/SafeMath.sol	5103643f24be2f2ec7e608cffc038c13e80cbfa9a8aac37ae1c1df139b15eb9c

7. Appendix C

Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
CodexRecord.sol	100.00%	100.00%	100.00%	100.00%
CodexRecordAccess.sol	100.00%	100.00%	100.00%	100.00%
CodexRecordCore.sol	100.00%	50.00%	100.00%	100.00%
CodexRecordFees.sol	92.86%	70.00%	100.00%	87.50%
CodexRecord Metadata.sol	100.00%	100.00%	100.00%	100.00%
CodexRecordProxy.sol	100.00%	83.33%	100.00%	100.00%
CodexStakeContract.sol	100.00%	100.00%	100.00%	100.00%
CodexStakeContractInterface.sol	100.00%	100.00%	100.00%	100.00%
ERC20/ERC20.sol	100.00%	100.00%	100.00%	100.00%
ERC20/ERC20Basic.sol	100.00%	100.00%	100.00%	100.00%
ERC165/ERC165.sol	100.00%	100.00%	100.00%	100.00%
ERC721/ERC721.sol	100.00%	100.00%	100.00%	100.00%
ERC721/ERC721Basic.sol	100.00%	100.00%	100.00%	100.00%
ERC721/ERC721BasicToken.sol	100.00%	92.31%	100.00%	100.00%
ERC721/ERC721Receiver.sol	100.00%	100.00%	100.00%	100.00%
ERC721/ERC721Token.sol	100.00%	100.00%	100.00%	100.00%
ERC900/ERC900.sol	100.00%	100.00%	100.00%	100.00%
ERC900/ERC900BasicStakeContract.sol	100.00%	80.00%	100.00%	100.00%
ERC900/ERC900Compoundin	100.00%	50.00%	100.00%	100.00%

gStakeContract.sol				
ERC900/ERC900CreditsStakeContract.sol	66.67%	33.33%	66.67%	66.67%
library/AddressUtils.sol	100.00%	100.00%	100.00%	100.00%
library/Debuggable.sol	100.00%	100.00%	100.00%	100.00%
library/DelayedOwnable.sol	100.00%	100.00%	100.00%	100.00%
library/DelayedPausable.sol	100.00%	100.00%	100.00%	100.00%
library/ProxyOwnable.sol	100.00%	100.00%	100.00%	100.00%
library/SafeMath.sol	100.00%	100.00%	100.00%	100.00%
All files	**97.85%**	**86.54%**	**97.83%**	**97.54%**