SMART CONTRACT AUDIT

ZOKYO.

June 12th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

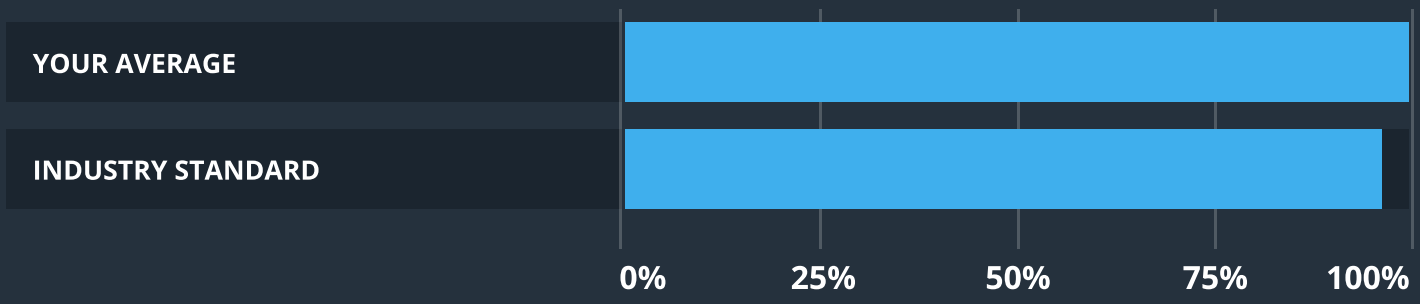
This document outlines the overall security of the iTrust smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the iTrust smart contract codebase for quality, security, and correctness.

Contract Status

There were critical issues found during the audit, but they were resolved by the iTrus team.

Testable Code



The testable code is 99%, which is above the industry standard of 95%.

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 Zokyo recommend that the iTrust team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the iTrust archive with source code.

SHA256 of the archive:
16dac7428bbc82794141fe7adbe35e9aa77b9090604e2f39ac0e846199cece93

Re-audit SHA256 of the archive:
dab0be1849bef5bebff33b236fca1aba1433955125b2ab307df408e488220fbe
Within the scope of this audit Zokyo auditors have reviewed the following contract(s):
iTrustVaultFactory.sol
Vault.sol
StakingData.sol
GovernanceDistribution.sol
Burn.sol
BaseContract.sol
RoundData.sol
StakeData.sol

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

- Implements and adheres to existing 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;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of iTrust smart contracts. To do so, the code is reviewed line-by-line by our 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.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were found several critical issues connected to the missing allowance and error in calculation - both errors may lead to issues with tokens logic. Also auditors have found problems in the storage usage, unused methods and local variables, with standard automatic tools issues' list violation, and with standard contracts used in the protocol.

There are several findings which have an impact on contracts performance, contract code style and further development.

Nevertheless, most of the findings were successfully resolved by the iTrust team.

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:

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.

COMPLETE ANALYSIS

CRITICAL | RESOLVED

Missing result assignment

Burn.sol, line 130. “_burnData[vaultAddress].totalBurned.add(toBurn);”
Total burned value is not updated - addition result is not placed to storage.

Recommendation:
fix the condition.

CRITICAL | RESOLVED

No check for allowance in transferFrom

Vault.sol, line 351
Missing checking sender's allowance in transferFrom method. Thus, anyone can spend other users' tokens

Recommendation:
check spender's allowance like in ERC20

HIGH | RESOLVED

Incorrect condition
StakingData.sol, line 416: Condition “i < 0” is always false for uint256. Review the functionality

Recommendation: fix the condition.

HIGH | RESOLVED

Incorrect contracts used

Some contracts inherit upgradeable contracts from OpenZeppelin contracts-upgradeable. So it should use all utility contracts from the upgradeable set. For now incorrect contracts from the “vanilla” set are used in a mixed case with upgradeable ones. Such approach can create collisions, affect the development and create unpredictable issues in the runtime.
Vault.sol: IERC20, SafeMath and ECDSA
StakingData.sol: SafeMath
Burn.sol: SafeMath
BaseContract.sol: SafeMath
RoundData.sol: SafeMath
StakeData.sol: SafeMath

Recommendation:
Fix the contracts

MEDIUM | UNRESOLVED

Unused storage variable
BaseContract.sol, _ReentrantCheck
Vault.sol, _ReentrantCheck
Variables are not used in the code.

Recommendation: Remove unused variables.

MEDIUM | RESOLVED

Solidity version update

The solidity version should be updated. Throughout the project (including interfaces).
Issue is classified as Medium, because it is included to the list of standard smart contracts' vulnerabilities. Currently used version (0.7.5) is not the last in the line, which contradicts the standard checklist.

Recommendation: You need to update the solidity version to the latest one in the branch - 0.7.6.

LOW | RESOLVED

Unused local variable

StakingData.sol, line 256.
Variable maxIteration is unused in the contract. Review the functionality or remove the variable.

Recommendation: Review the functionality or remove the variable.

LOW | RESOLVED

Unused internal constants

Constants STATUS_DEFAULT and STATUS_CANCELED declared in Burn.sol are never used in Burn.sol

Recommendation: review the functionality or remove constants.

LOW | RESOLVED

Unused method for paused status

Vault.sol, 437 _ifNotPaused() is never used.
So it makes function isPaused() from iTrustVaultFactory.sol unused as well.
Also these method are actual duplicates for isActiveVault() method, so isPaused() can be safely removed.

Recommendation:
Remove unused method.

LOW | UNRESOLVED

Re-use local variable

Vault.sol, line 188, “require(msg.value == _AdminFee)”
Variable adminFee was added for gas savings and can be re-used in the expression instead of _AdminFee.

Recommendation:
Re-use local variable

LOW | RESOLVED

Ignored return value

ITrustVaultFactory.sol, createVault() ignores return value by stakingDataContract.addVault()

Recommendation:
consider adding require statement.

LOW | RESOLVED

Unused functions

ITrustVaultFactory.sol: isPaused, _onlyAdmin
Burn.sol: _vaultAddress, _getStartOfDayTimeStamp, validateFactory, _valueCheck
GovernanceDistribution.sol: _getStartOfDayTimeStamp
StakingData.sol: getTotalSupplyForBlock, getHoldingsForIndexAndBlock, getNumberOfStakingAddresses

Recommendation:
Remove unused functions

LOW | UNRESOLVED

Useless boolean return

StakingData.sol, endRound(), addVault() - always return true, which makes the return value useless.

Recommendation:
remove return value.

LOW | UNRESOLVED

Boolean equality comparison

-require(bool)(_AdminList[newAddress] == false) (iTrustVaultFactory.sol#70)
-require(bool)(_TrustedSigners[newAddress] == false) (iTrustVaultFactory.sol#83)
-_TrustedSigners[account] == true (iTrustVaultFactory.sol#88)
-_AdminList[account] == true (iTrustVaultFactory.sol#144)
-_VaultStatus[msg.sender] == false (iTrustVaultFactory.sol#148)
-_VaultStatus[vaultAddress] == true (iTrustVaultFactory.sol#152)
-require(bool,string)(_AdminList[msg.sender] == true) (iTrustVaultFactory.sol#156-159)
-require(bool,string)(_AdminList[msg.sender] == true) (iTrustVaultFactory.sol#33)
-require(bool)(vaultFactory.isPaused() == false) (vaults\Vault.sol#439)

Recommendation:
Use boolean values directly without equality comparison

LOW | UNRESOLVED

Use storage pointer

StakingData.sol, line 308, _getAllAcountUnstakesForAddress()
StakingData.sol, line 287, _getAccountStakesForAddress()
StakingData.sol, line 225, _getRoundRewardsForAddress()
RoundData.sol, line 15, endRound()
Burn.sol, line 64, getCurrentBurnData()
Burn.sol, line 77, startBurn()
Burn.sol, line 140, endBurn()
Consider usage of storage pointer to mapping member in order to get gas savings since the function has several calls to this data.

Recommendation:
Use storage pointer.

INFORMATIONAL | RESOLVED

Consider usage of exponential notation

There are several places with literals with too many digits. Consider usage of constants for them with exponential notation. It will increase the readability of the code and decrease the chance of the typo error in the number of digits.
div(1000000000000000000) (vaults\Burn.sol#115)
div(1000000000000000000) (vaults\StakingData.sol#367)

Recommendation:
Use “snake” literals form or use exponential notation.

INFORMATIONAL | UNRESOLVED

Use standard ReentrancyGuard

Throughout the project the variable _Locked together with _nonReentrant() function are used for the reentrancy prevention, though, for the safety of further development it is recommended to use standard ReentrancyGuard with modifier. It will increase the overall code quality.

Recommendation:
Use standard ReentrancyGuard.

	GovernanceDistribution	StakingData, BaseContract	Vault
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo team

As part of our work assisting iTrust in verifying the correctness of their contract code, our team was responsible for writing integration tests using Truffle testing framework.

Tests were based on the functionality of the code, as well as review of the iTrust contract requirements for details about issuance amounts and how the system handles these.

```
Testing Factory
  Test initializer
    ✓ Should not create factory if admin is zero (789ms)
  Test admin functions
    ✓ Should not let not admin call admin functions (85ms)
    ✓ Should pause and unpause vault (669ms)
    ✓ Should add admin correctly (222ms)
    ✓ Should revoke admin correctly (250ms)
    ✓ Should not add trusted signer if already a signer (456ms)
    ✓ Should update addresses of contracts (1293ms)
    ✓ Should return correct burnData address (217ms)
    ✓ Should revoke trusted signer correctly (981ms)

Testing Vault
  Test setters
    ✓ Should set new admin fee (676ms)
    ✓ Should set new comission (588ms)
    ✓ Should set treasury (286ms)
```

```
Test deposit/unstake/withdraw/burn/endRound functions
  ✓ Should deposit nxm (1309ms)
  ✓ Should not deposit zero (153ms)
  ✓ Should revert depositNXM if tokens were not approved (760ms)
  ✓ Should revert depositNXM if stake creation failed (929ms)
  ✓ Should deposit wnxm (1067ms)
  ✓ Should revert depositWNXM if tokens were not approved (791ms)
  ✓ Should revert depositWNXM if stake creation failed (863ms)
  ✓ Should start unstake and withdraw unstaking (2646ms)
  ✓ Should revert startUnstake if value exceeds total staking (975ms)
  ✓ Should revert in authorizeUnstake if accounts length > 10 (436ms)
  ✓ Should not authorize unstake (1710ms)
  ✓ Should revert in withdraw if amount > balance (199ms)
  ✓ Should revert withdraw if nothing to withdraw (802ms)
  ✓ Should not withdraw adminFee in startUnstake (1146ms)
  ✓ Should revert startUnstake if msg.value < adminFee (745ms)
  ✓ Should withdraw rewards (2231ms)
  ✓ Should not let not signer approve withdraw rewards (1980ms)
  ✓ Should revert withdraw rewards if rewards are zero (2002ms)
  ✓ Should not let use the same nonce (2538ms)
  ✓ Should burn tokens (1434ms)
  ✓ Should not burn tokens if sender is not a valid burner (950ms)
  ✓ Should not burn tokens if amount is zero (1002ms)
  ✓ Should not burn tokens if amount > total staked (2384ms)
  ✓ Should revert endRound if tokens and amounts arrays length mismatch (230ms)
  ✓ Should revert endRound if call to stakingData failed (660ms)
  ✓ Should revert endRound if token address is zero (3160ms)
  ✓ Should not withdraw comission in endRound (3643ms)
  ✓ Should revert endRound if tokens were not approved (5609ms)
```

```
  Test getters
    ✓ Should get correct round data (4443ms)
    ✓ Should get correct reward round data (3908ms)
    ✓ Should get account stakes (2810ms)
    ✓ Should get all accounts unstakes (4919ms)
    ✓ Should get account unstaked total and total unstaked wnxm (4388ms)
    ✓ Should check admin (425ms)
    ✓ Should calculate rewards (4096ms)
    ✓ Should calculate total unstakings for block range (12223ms)
  Test transfer
    ✓ Should transfer tokens (9444ms)
    ✓ Should transfer from tokens (7774ms)
    ✓ Should revert transfer if remove stake failed (3024ms)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	100	92.86	100	100	
iTrustVaultFactory.sol	100	92.86	100	100	
contracts\vaults\	99.38	97.73	99.15	99.38	
BaseContract.sol	100	100	100	100	
Burn.sol	100	100	100	100	
GovernanceDistribution.sol	100	95	100	100	
StakingData.sol	98.1	95.96	97.73	98.11	
Vault.sol	100	85.14	100	100	
contracts\vaults\StakingDataController\	96.77	82.14	100	95.96	
RoundData.sol	100	100	100	100	
StakeData.sol	96	87.5	100	95	
All files	99.04	95.96	99.37	98.9	

We are grateful to have been given the opportunity to work with the iTrust team.

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.

Zokyo's Security Team recommends that the iTrust team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.