OIN

SMART CONTRACT AUDIT

ZOKYO.

July 6th, 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

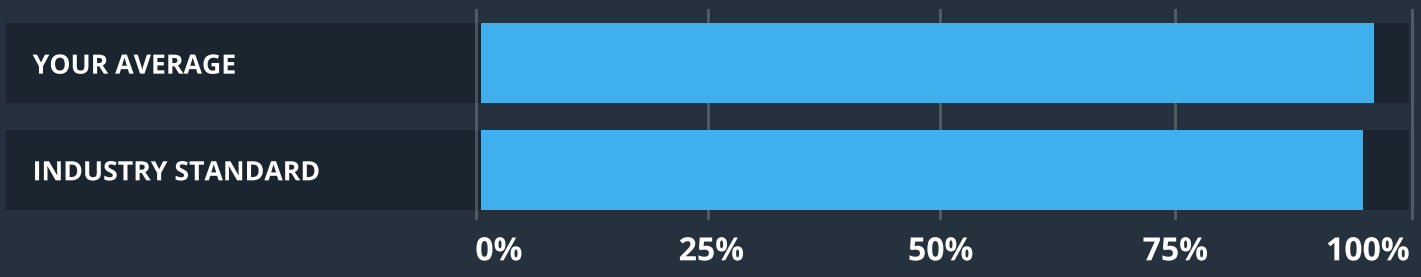
This document outlines the overall security of the OIN smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were no critical issues found during the audit.

Testable Code



The testable code is 96.18%, 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 OIN 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 OIN repository – https://github.com/oinfinance/OINDAO2.1

Last commit – 3901db151c10c7e7d99124fb28b77f560b9c01aa

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 OIN 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

Generally, the contracts provided for an audit are well written and structured. All the findings within the auditing process are presented in this document.

There were no critical issues found during the audit. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner. The findings during the audit have no impact on contract performance or security, so it is fully production-ready.

Taking into account the fact that mostly all of the issues were resolved and evaluating the contracts from the operational and security standpoints, we can give the score of 94% .

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 ability of the contract to compile or operate in a significant way.

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.

MANUAL REVIEW

HIGH | UNRESOLVED

Solidity files are not pointing to a specific Solidity compiler version.

Recommendation:
Use specific Solidity compiler version.

MEDIUM | RESOLVED

Contract 'OinStake', function getParamsAddr() will return incorrect values for the params. it will return zero values for all returned variables.

Recommendation:
Change the function getParamsAddr() in the next way:

```
function getParamsAddr()
    public
    view
    returns (
      address ,
      address ,
      address ,
      address
    )
 {
    return (address(esm), address(params), address(orcl), address(feeOrcl));
 }
```

LOW | UNRESOLVED

Contract ‘Oracle’, function peek() is useless cause the val variable is public:

```
function peek() public view returns (uint256) {
    return val;
}
```

LOW | RESOLVED

Contract ‘OinStake’ and ‘Oracle’, in functions extractReward(), injectReward() and poke() missing require that will check if the amount (price for poke() function) is not zero value.

LOW | RESOLVED

Contract ‘Esm’. Missed require in function setupTokenStake() that will check if _tokenStake address is not zero.

INFORMATIONAL | UNRESOLVED

Contract ‘OinStake’ function updateIndex(). In line #433 not needed initializing:

```
rewardCoins[i].index = rewardCoins[i].index; // line 433
```

INFORMATIONAL | UNRESOLVED

Contract 'Dparam' function isUpperLimit based on the name and notice should return true if value is more then upper limit but it returns false.

Recommendation:
Invert a logical expression or change name of function.

INFORMATIONAL | RESOLVED

In contract 'OInStake' there is a misleading comment that says there is a parameter 'staker' in the struct FundsFeeState, but none really.

Recommendation:
Remove unnecessary comment:

```
/**
 * @notice fundsFee array
 * @param staker fundsFee
 * @param block update blockNumber
 */

struct FundsFeeState {
    uint256 block;
}
```

INFORMATIONAL | RESOLVED

The function InitRewardCoin() in contract OinStake just called in the constructor to set default reward coin, but the whiteListed address can call it what is not needed in practical use. So it is not needed to make this function public.

Recommendation:
To optimize gas usage make InitRewardCoin() function private or internal.

INFORMATIONAL | RESOLVED

Unused variable in contract OinStake in line #115 uint256 constant ONE = 10**8; // line 115.

INFORMATIONAL | RESOLVED

Contract 'OinStake', function getInputToken(). The mislead comment in notice section.

Recommendation:
Change the comment according to the current function goal.

```
/**
 * @notice Get the number of debt by the 'account'
 * @param coinAmount The amount that staker want to get stableToken
 * @return The amount that staker want to transfer token.
 */

function getInputToken(uint256 coinAmount)
    public
    view
    returns (uint256 tokenAmount)
{
    tokenAmount = coinAmount.mul(param.stakeRate()).div(1e8);
}
```

INFORMATIONAL | RESOLVED

Contract 'OinStake'. Mislead comments for function debtOf(), _getFundsFee(), getFundsFee(), _judgePledgeRate(). The parameters 'account' and 'staker' are not the 'token address', but 'user address'.

```
* @param account token address
* @param  staker token address
```

INFORMATIONAL | RESOLVED

Contract 'OinStake' function _getFundsFee(), there are two unused variables in 378 and 379 lines.

```
FundsFeeState storage FundsFeeState = fundsFeeStates[staker]; //line 378
uint256 fundsRate = params.fundsRate(); // line 379
```

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

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

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

Code Coverage

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	96.18	71.43	97.32	95.98	
Context.sol	100.00	100.00	100.00	100.00	
Dparam.sol	100.00	100.00	100.00	100.00	
ERC20.sol	100.00	75.00	100.00	100.00	
ERC20Detailed.sol	100.00	100.00	100.00	100.00	
Esm.sol	100.00	75.00	100.00	100.00	
Math.sol	100.00	100.00	100.00	100.00	
OinStake.sol	95.14	66.67	100.00	95.16	... 522, 543, 599
Oracle.sol	92.31	83.33	100.00	92.31	65
Owned.sol	100.00	83.33	100.00	100.00	
Pausable.sol	100.00	87.50	100.00	100.00	
SafeMath.sol	84.21	58.33	66.67	80.00	144, 165, 166, 173
State.sol	100.00	75.00	100.00	100.00	
USDOToken.sol	100.00	100.00	100.00	100.00	

WhiteList.sol	100.00	85.00	100.00	100.00
cToken.sol	100.00	100.00	100.00	100.00
All files	**96.18**	**71.43**	**97.32**	**95.98**

Test Results

Contract: Dparam

Setup
- ✓ should create contracts (10137ms)

On success
- ✓ should set funds rate (388ms)
- ✓ should set liquidation line (324ms)
- ✓ should set total coin (510ms)
- ✓ should set coin upper limit (377ms)
- ✓ should set claim requirment (259ms)
- ✓ should set cost (395ms)
- ✓ should call isUpperLimit (253ms)
- ✓ should call isLiquidation (375ms)
- ✓ should call removeWhiter (586ms)

On reverts
- ✓ should NOT set coin upper limit (714ms)
- ✓ should NOT appendWhiter (1209ms)

Contract: Esm

Setup
- ✓ should create contracts (3462ms)

On success
- ✓ should setup token stake (528ms)
- ✓ should pause deposit (874ms)
- ✓ should pause withdraw (690ms)
- ✓ should open deposit (336ms)
- ✓ should open withdraw (464ms)
- ✓ should pause generate (457ms)
- ✓ should open generate (352ms)
- ✓ should pause payback (607ms)
- ✓ should open payback (363ms)
- ✓ should shutdown (884ms)

On reverts
- ✓ should NOT shtdown (232ms)
- ✓ should NOT set oracle price (3671ms)

Contract: Oinstake

Setup
- ✓ should create contracts (3907ms)
- ✓ should call setup (10449ms)
- ✓ should set price (2467ms)
- ✓ should add new reward token (2257ms)

On Success

On deposit
- ✓ should deposit (16960ms)
- ✓ should generate coins [1] (2612ms)
- ✓ should withdraw (1864ms)
- ✓ should set new reward speed (991ms)
- ✓ should payback (2389ms)
- ✓ should generate coins [2] (2264ms)

On exit
- ✓ should inject rewards (1220ms)
- ✓ should call getHolderReward (175ms)
- ✓ should claim rewards (1652ms)
- ✓ should extract rewards (1198ms)
- ✓ should call oRedeem (2736ms)

On params
- ✓ should call debtOf (141ms)
- ✓ should call getInputToken (125ms)
- ✓ should call getFundsFee (156ms)
- ✓ should call getStakerStateArray (297ms)

On reverts
- ✓ should deploy new instanse (14463ms)
- ✓ should NOT setup (1348ms)
- ✓ should NOT add reward coin (2281ms)
- ✓ should NOT deposit (11766ms)
- ✓ should NOT withdraw (1105ms)

Contract: Math

Setup

```
    ✓ should create contracts (840ms)
  On success
    ✓ should do math operations (498ms)

Contract: Tokens
  Setup
    ✓ should create contracts (2503ms)
  cToken
    On success
      ✓ should mint (664ms)
      ✓ should burn (295ms)
      ✓ should set newassociated contract (587ms)
      ✓ should call data from ERC20Detailed (374ms)
    On reverts
      ✓ should NOT mint (238ms)
    On ownable
      ✓ should change owner (1098ms)
    On pausable
      ✓ should pause contract (1350ms)
  USDOToken
    On success
      ✓ should mint (706ms)
      ✓ should burn (413ms)
      ✓ should set newassociated contract (567ms)
    On reverts
      ✓ should NOT mint (303ms)
  ERC20
    On success
      ✓ should mint (631ms)
      ✓ should transfer (2961ms)
      ✓ should burn (600ms)
      ✓ should call totalSupply (101ms)
      ✓ should call msgData (109ms)
    On reverts
      ✓ should NOT trasfer (1051ms)
      ✓ should NOT mint (351ms)

70 passing (2m)
```

We are grateful to have been given the opportunity to work with the OIN 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 OIN team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.