IX SWAP

SMART CONTRACT AUDIT

ZOKYO.

September 2nd, 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

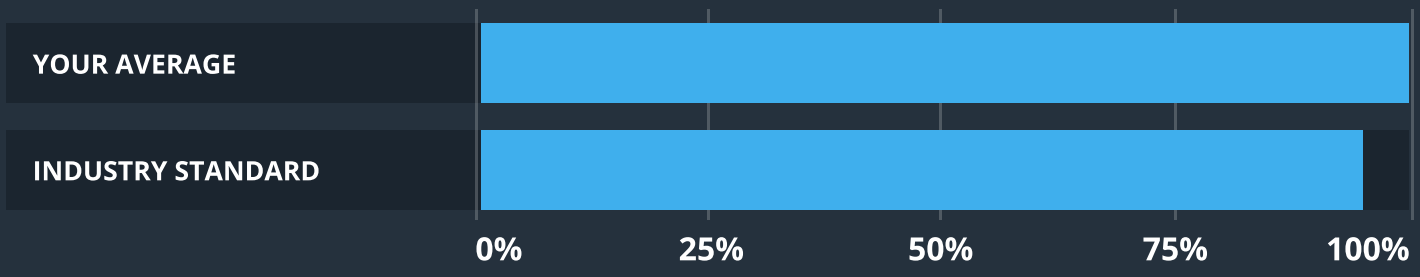
This document outlines the overall security of the IXS smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the IXS 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 100%, 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 IXS 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 IXS repository.

Repository (archive hash):
015549c0b92df55ccc4ac0afdbd2ec867e7ab791

Last commit (archive hash):
649009b51ce9ceb2bed450f54e8f118a3e3da553

Contracts:

- IxsToken.sol
- IxsGovernanceToken.sol
- IxsStakeBank.sol
- IxsReturningStakeBank.sol
- IXSVestedDistribution.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 IXS 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.

SUMMARY

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.

During the conducted investigation we found 2 issues with low severity. Both of them are resolved and bear no risk for the contract owner or end user. Zokyo team can state that the contracts are secure and bear no operational risk.

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.

COMPLETE ANALYSIS

Correcting the require statements to be more precise

LOW | NOT VALID

Since the transactions would fail if the require condition is not satisfied, So it is recommended to add a statement which justifies the reason for failing of the transaction.

Recommendation:

1) In IXSVestedDistribution.sol file, in constructor(), replace

```
require(_owner != address(0), "IxsStakeBank: INVALID_OWNER")
```

with

```
require(_owner != address(0), "IXSVestedDistribution: INVALID_OWNER")
```

2) In IxsReturningStakeBank.sol, in stakeFor(), replace

```
require(MintableBurnableIERC20(returnToken).mint(user, amount), 'IxsStakeBank: MINT_FAILED')
```

with

```
require(MintableBurnableIERC20(returnToken).mint(user, amount), IxsReturningStakeBank: MINT_FAILED')
```

Lock pragma to specific version

LOW | RESOLVED

In IxsToken.sol file, lock the pragma to a specific version, since not all the EVM compiler versions support all the features, especially the latest one's which are kind of beta versions, So the intended behaviour written in code might not be executed as expected.

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.

	IxsToken	IxsStakeBank	IXSVestedDistribution
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

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting IXS 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 IXS 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\	100.00	93.14	100.00	100.00	
IXSVestedDistribution.sol	100.00	92.86	100.00	100.00	
IxsGovernanceToken.sol	100.00	100.00	100.00	100.00	
IxsReturningStakeBank.sol	100.00	83.33	100.00	100.00	
IxsStakeBank.sol	100.00	93.75	100.00	100.00	
IxsToken.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**93.14**	**100.00**	**100.00**	

Test Results

```
Contract: IxsStakeBank
  Initialization
    ✓ cannot deploy with zero owner address (747ms)
    ✓ cannot deploy with zero token address (304ms)
  Staking
```

✓ should stake tokens correctly (1908ms)
✓ cannot stake with low allowance (1335ms)
✓ should unstake tokens correctly (498ms)
✓ cannot unstake if amount of tokens to unstake more than total staked (297ms)
✓ should return staking history for address correctly (328ms)
✓ should return total staked correctly (258ms)
✓ should return total staked for address correctly (218ms)
✓ should return first staked block for address correctly (201ms)
✓ should return last staked block for address correctly (880ms)
✓ should return total staked for address at block correctly (309ms)
✓ should return total staked for address at time correctly (266ms)
✓ should return the total tokens staked at block timestamp correctly (142ms)
✓ should return minimal amount of tokens staked between block timestamps for address correctly (6243ms)

Contract: IxsReturningStakeBank
Initialization
✓ should deploy with correct returnToken address (91ms)
✓ cannot deploy with zero returnToken address (376ms)
✓ cannot deploy with the same tokens (475ms)
Staking
✓ should stake tokens correctly (1572ms)
✓ should unstake tokens correctly (726ms)

Contract: IxsToken
Initialization
✓ should deploy with correct token name (127ms)
✓ should deploy with correct token symbol (98ms)
✓ should deploy with correct decimals (92ms)
✓ should deploy with correct total supply (130ms)
✓ cannot deploy with zero treasury address (543ms)
Minting
✓ should mint tokens correctly (287ms)
✓ cannot mint more tokens than max supply (283ms)

Contract: IxsGovernanceToken
Initialization
✓ should deploy with correct token name (134ms)

```
    ✓ should deploy with correct token symbol (114ms)
    ✓ should deploy with correct decimals (82ms)
    ✓ cannot deploy with zero owner address (367ms)

Contract: IXSVestedDistribution
  Initialization
    ✓ should deploy with correct token address (51ms)
    ✓ cannot deploy with zero token address (512ms)
    ✓ cannot deploy with zero owner address (407ms)
  Distribute
    ✓ should distribute tokens correctly (808ms)
    ✓ cannot distribute if investor is vesting (199ms)
    ✓ cannot distribute with zero amount (346ms)
    ✓ cannot distribute with zero vesting period (247ms)
    ✓ cannot distribute if vesting duration less than cliff (168ms)
    ✓ cannot distribute with insufficient supply (766ms)
  AddFunds
    ✓ should add funds correctly (445ms)
    ✓ cannot add funds with zero amount (301ms)
    ✓ cannot add funds for unknown investor (195ms)
    ✓ cannot add funds with insufficient supply (264ms)
  Claim
    ✓ should return available claim correctly (661ms)
    ✓ should batchClaimFor correctly (1952ms)
    ✓ should claimFor correctly (1315ms)
  Other functions
    ✓ should return payouts correctly (457ms)
    ✓ cannot check vesting with zero investor address (88ms)
    ✓ cannot return details for unknown investor (265ms)

50 passing (38s)
```

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

ZOKYO.