ixo Smart Contract Audit

1. Introduction

iosiro was commissioned by ixo Foundation to conduct an audit on their token and protocol smart contracts. The audit was performed between 10 September 2018 and 20 September 2018.

This report is organized into the following sections.

- **Section 2 - Executive Summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit Details:** A description of the scope and methodology of the audit.
- **Section 4 - Design Specification:** An outline of the intended functionality of the smart contracts.
- **Section 5 - Detailed Findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to understand the risk exposure of the smart contracts, and as a guide to improve the security posture of the smart contracts by remediating the issues that were identified. The results of this audit are only a reflection of the source code reviewed at the time of the audit and of the source code that was determined to be in-scope.

2. Executive Summary

This report presents the findings of an audit performed by iosiro on the ixo token and protocol smart contracts. The purpose of the audit was to achieve the following.

- Ensure that the smart contracts functioned as intended.
- Identify potential security flaws.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered very high risk with regards to cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. There are a number of techniques that can help to achieve this, some of which are described below.

- Security should be integrated into the development lifecycle.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed wherever possible.

At the conclusion of the audit, only informational findings and design recommendations remained open. These included minor deviations from best practice, unnecessary variable declaration, and instances where code could be simplified.

Despite the findings, the code was of a relatively high standard. This was evident in the use of modularized code, as well as making use of commonly used libraries where reasonable.

The risk posed by the smart contracts can be further mitigated by using the following controls prior to releasing the contracts to a production environment.

- Use a public bug bounty program to identify security vulnerabilities.
- Perform additional audits using different teams.
- Extending the test suite coverage.

3. Audit Details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from any other files are considered to be out-of-scope.

3.1.1 ixo Protocol Smart Contracts

Project Name: ixo-solidity
Commits: 417c40f, f7ca254

3.2 Methodology

A variety of techniques were used to perform the audit, these are outlined below.

3.2.1 Dynamic Analysis

The contracts were compiled, deployed, and tested using both Truffle tests and manually on a local test network. A number of pre-existing tests were included in the project.

3.2.2 Automated Analysis

Tools were used to automatically detect the presence of potential vulnerabilities, such as reentrancy, timestamp dependency bugs, transaction-ordering dependency bugs, and so on. Static analysis was conducted using Mythril and Oyente. Additional tools,

such as the Remix IDE, compilation output and linters were used to identify potential security flaws.

3.2.3 Code Review

Source code was manually reviewed to identify potential security flaws. This type of analysis is useful for detecting business logic flaws and edge-cases that may not be detected through dynamic or static analysis.

3.3 Risk Ratings

Each Issue identified during the audit is assigned a risk rating. The rating is dependent on the criteria outlined below..

- **High Risk** - The issue could result in a loss of funds for the contract owner or users.
- **Medium Risk** - The issue results in the code specification operating incorrectly.
- **Low Risk** - A best practice or design issue that could affect the security standard of the contract.
- **Informational** - The issue addresses a lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed** - The issue was identified during the audit and has since been addressed to a satisfactory level to remove the risk that it posed.

---

4. Design Specification

The following section outlines the intended functionality of the smart contracts.

4.1 ixo Token

The ixo protocol token is described below.

ERC-20 Token

The token should implement the ERC-20 standard.

Field	Value
Symbol	IXO
Name	IXO Token
Decimals	8
Supply Cap	10,000,000,000

Mintable

Tokens may be minted until the supply cap is reached. It should only be possible for the designated minter address to mint new tokens.

4.2 ixo Protocol Smart Contracts

The functionality of the ixo protocol smart contracts is described below.

Project Token Wallet

Project wallets should support storing and transferring any single ERC-20 compliant token.

Project Wallet Registry

Project wallets should be recorded according to their projectDID in a central project wallet registry contract. Additionally, it should be possible for the registry to instantiate new wallets associated with a projectDID.

Authorisation

Transfer requests can be initiated by designated members set by the contract owner. A transfer must only be triggered once a set quorum of members is reached.

Wallet Authoriser

A wallet authoriser contract is used to allow the authorisation scheme to be changed for a wallet by the contract owner. This means that while the current configuration requires a certain quorum of members to be reached before a transfer is approved, different requirements could be implemented at a later stage.

5. Detailed Findings

The following section includes in depth descriptions of the findings of the audit.

5.1 High Risk

No high risk issues were present at the conclusion of the audit.

5.2 Medium Risk

No medium risk issues were present at the conclusion of the audit.

5.3 Low Risk

No low risk issues were present at the conclusion of the audit.

5.4 Informational

5.4.1 Design Comments

Actions to improve the functionality and readability of the codebase are outlined below.

Calls to `validate(...)` should be reverted after action is triggered

AuthContract.sol: Lines 66-68

It was found that the `if` statement used to check whether an action had already been triggered returned `true` if the action had already been triggered. Although strictly not a security issue, it would be more appropriate to simply use `require` and have the function revert in the case that the action had already been triggered. This will reduce the gas cost of subsequent calls to `validate(...)` from members after the action has already been triggered and return a more logical result considering no transfer of tokens would take place. An example of the code is below:

```
require(!actions[_tx].triggered, "Already triggered");
```

Return type specified with no return value in function body

ProjectWalletAuthoriser.sol: Line 18

It was found that the `setAuthoriser(..)` function explicitly defined a return type of `bool` which was unused in the function body. Although strictly not a security or functional issue, it should be removed in order to adhere to best practice.

Return type specified for `transfer(...)` functions with no return value in function body

ProjectWalletAuthoriser.sol: Line 35, BasicProjectWallet.sol: Line 42

In several cases, the `transfer(...)` function was specified with the return type `bool`, however the functions in question did not return the result of the transfer. It is recommended that the functions in question return the result of the transfer to adhere to best practice.

`Confirmed` and `Triggered` events should index the transaction ID

AuthContract.sol: Lines 25-26

It was found that the logging events `Confirmed` and `Triggered` did not use the keyword `indexed` on the `id` parameter. Indexing the `id` parameter would allow for the transaction events to be searched and filtered by users, providing better transparency and logging. It is thus recommended that the `indexed` keyword be used.

Incorrect `require` validation failure messages

AuthContract.sol: Lines 81-84

It was found that several validation error messages had improper messages associated with them. Although strictly not a security or functional issue, it should be resolved to provide better feedback in cases where validation fails.

Use of implicit integer sizes in AuthContract and ixoERC20Token contract implementations

AuthContract.sol and ixoERC20Token.sol

It was found that several properties and functions contained implicit `uint` type declarations. It is suggested that these `uint` type declarations be replaced by the

`uint256` type declaration instead in order to adhere to best practice.

Inexact solidity compiler version used

The pragma version was not fixed to a specific version, as it specified `^0.4.24`, which would result in using the highest non-breaking version (highest version below `0.5.0`). According to best practice, where possible, all contracts should use the same compiler version, which should be fixed to a specific version. This helps to ensure that contracts do not accidentally get deployed using an alternative compiler, which may pose the risk of unidentified bugs. An explicit version also helps with code reuse, as users would be able to see the author's intended compiler version. It is recommended that the pragma version is changed to a fixed value, for example `0.4.24`.

5.5 Closed

5.5.1 Multiple High Risk Authorisation Issues (High)

AuthContract.sol: lines 58 - 85

Description

Several high risk security issues were identified in the `validate(...)` function. These are outlined below.

Multiple Confirms Permitted Per Member

It was possible for a single member to call `validate(...)` several times in order to reach the required quorum for a transaction. This vulnerability resulted in any one member being able to trigger an event without requiring interaction from any other members. The issue originated from there being no check for whether the member had already confirmed the action.

It is recommended that logic is added to prevent a member from being able to call the function more than once. This could be implemented by using a mapping of addresses to booleans, which stores whether a given member has called the `validate(...)` function. This mapping should then be referenced to determine whether a member is permitted to call the function.

Possible to Replay Transactions

A vulnerability was identified that allowed an action to be called repeatedly once the quorum had been reached. While line 88 of AuthContract.sol set

`actions[_tx].triggered` to `true`, line 74 of AuthContract.sol set the value back to false, effectively resetting the status of the action. Once the action had been triggered, it would simply require calling the `validate(...)` function again to reset the triggered status and perform the action again.

It is recommended that an assertion is performed to verify that the triggered status is false. For example:

```
require(!actions[_tx].triggered, "Already triggered");
```

Additionally, line 79 should be modified to require exactly the number of confirmations, instead of greater or equal to the number of confirmations.

Last Member to Confirm Determines Final State

As the values of the action were set (lines 70 - 75) on each call to `validate(...)`, it would be possible for the last member calling the function to modify all of the values of the action. This would allow a malicious member to wait for `quorum-1` confirmations before submitting their confirmation with their desired values.

It is recommended that the logic of the function is modified to only allow the values for an action to be set when the action is created (`confirmations == 0`).

Implemented Action

Fixed in f7ca254.