AMO Crowdsale Smart Contract Audit

1. Introduction

iosiro was commissioned by Penta Security Systems to conduct an audit on their token and crowdsale smart contracts for their AMO ICO. The audit was performed between 18 April 2018 and 23 April 2018. On 26 April 2018, a review was conducted to confirm that issues raised in this report had been sufficiently addressed.

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 AMO token and crowdsale 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.

By the conclusion of the audit, only one informational issue remained open. This issue related to design recommendations that would more strictly implement expected behaviour and best practice recommendations. During the initial audit, one medium risk vulnerability was identified that allowed accounts to withdraw locked tokens. This issue was fixed at the time of the review.

The code was generally well designed and clearly written. It separated token and crowdsale logic and made use of commonly used libraries, where reasonable. A comprehensive test suite was also provided by the development team.

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.

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 AMO-Project

Project Name: AMO-Contracts
Commit: 26b5b22
Files: AMOCoin.sol, AMOCoinSale.sol

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 AMO Coin

The AMO Coin token is described below.

ERC20 Token

The token implements the ERC20 standard.

Field	Value
Symbol	AMO
Name	AMO Coin
Decimals	18
Total Supply	20,000,000,000

Token Allocations

- Crowdsale - 10,000,000,000 tokens
- Admin (Team) - 10,000,000,000 tokens

Lock

It is possible to place a lock on an address. This prohibits a token holder from withdrawing more than a specified number of tokens.

Pausable

It is possible to stop the transfer of tokens entirely, except for the admin account and token sale addresses. The default state is paused.

Burnable

It is possible to remove tokens from the total supply.

4.2 AMO Crowdsale

The AMO Coin crowdsale is described below.

Rounds

There are three distinct rounds:

- Early Investment - Tokens can be purchased through the crowdsale or sent by the admin account directly to participants without requiring ether in exchange. This can be used for offchain contributions.
- Presale - Tokens can be purchased in exchange for ether at the specified exchange rate.
- Crowdsale - Tokens can be purchased in exchange for ether at the specified exchange rate.

Stages

Each round has three stages these are described below.

- Setup - The round details (i.e. minimum contribution amount, maximum contribution amount, hard cap, and exchange rate per round) can be set at this stage.
- Started - It is possible to purchase tokens during this stage.
- Ended - At this stage, it is possible to proceed to the next round.

Whitelist

Participants need to be whitelisted before being able to contribute funds to the crowdsale.

Allocations

During the early investment round, it is possible for the crowdsale owner to send tokens to participants without requiring ether in exchange.

Defaults

- Hard cap - 12,000 ether
- AMO to ETH rate - 200,000 AMO/ETH
- Minimum contribution amount - 0.1 ETH

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

General

The following describes possible actions to improve the functionality and readability of the codebase.

Round times unenforced

It was possible to call the endSale function, ending the crowdsale, at any stage during the `Started` stage. It is recommended that assertions are used to ensure that now is larger than `endTime` or that the `weiRaised` round variable is used to determine whether the round hard cap has been reached. In this way, participants can be assured that the crowdsale will last the expected duration.

Possible to restart rounds

It was possible to call the `setUpSale` function with the same round multiple times. While this does not impact on the functionality of the code, it can lead to unintuitive outcomes. For example, it may be possible to have multiple presale rounds. Unless this is desired functionality, it is recommended that the round is automatically incremented when `endSale` is called, ensuring that only the expected rounds are used.

Private state variables

A number of state variable were found to have a `private` visibility set. As all information on the Ethereum blockchain is visible, it is still possible to access variables marked as private. The `private` visibility simply prohibits other contracts from accessing the variable. It is recommended that the following changes are made:

- AMOCoin.sol: lines 15, 70 should be made public.
- AMOCoinSale.sol: line 29 should be made public

From a developer's perspective, marking a variable as private complicates the process of ensuring that variables have been correctly assigned, and offers little to no security benefit.

No emit keyword

The emit keyword was found to be missing before events. This style has been deprecated, so it is recommended that the `emit` keyword is added before each event, e.g. `emit Transfer(...)`.

Inexact solidity compiler version used

The pragma version was not fixed to a specific version, as it specified `^0.4.18`, 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 `0.4.18`.

Timestamp

The `now` keyword (an alias for `block.timestamp`) is used to determine the current time. This value can be marginally affected by miners (by up to 900 seconds), so a common best practice is to rather use block.number to determine the time. However, the risk posed in this circumstance is inconsequential, and it is simply listed for completeness. No action is necessary.

5.5 Closed

5.5.1 Token Lock Bypass (Medium)

AMOCoin.sol: lines 153 - 160

Description

It was found that the token lock enforced by the smart contract could be bypassed, allowing an address to transfer all of their tokens at any stage. This vulnerability stemmed from the fact that the `transferFrom` function did have the same modifiers as the transfer function.

In order to exploit this attack, an account with locked tokens would simply need to call the approve function, allowing an address that they controlled to withdraw tokens on their behalf. As the `onlyAllowedAmount` modifier was not used on the `transferFrom` function, the tokens would be transferred out of the locked address into one that had no lock.

Remedial Action

It is recommended that the `onlyAllowedAmount(from, value)` modifier is used in the transferFrom function to correctly lock the funds. While the transfer function uses `onlyAllowedAmount(msg.sender, value)`, it is important that in the case of `transferFrom` that the from value is used rather than `msg.sender`, as the `msg.sender` would appear to be an unlocked account.

Implemented Action

Fixed in 2ccc569.

5.5.2 Design Comments (Informational)

General

Misleading permissions table

AMOCoin.sol contained comments that outlined permissions for different addresses. However, the table did not accurately reflect the permissions. The comments suggested that the following permissions were enforced:

transferEnabled: false	Owner	Admin	SalesContract	Others
transfer	x	x	x	x
transferFrom	x	v	v	x

transferEnabled: true	Owner	Admin	SalesContract	Others
transfer	v	x	x	v
transferFrom	v	x	x	v

While the table accurately reflected which functions would need to be used by each address at the relevant stage, it did not reflect the actual permissions of the users. The permissions table should have been:

transferEnabled: false	Owner	Admin	SalesContract	Others
transfer	x	v	v	x
transferFrom	x	v	v	x

transferEnabled: true	Owner	Admin	SalesContract	Others
transfer	v	v	v	v
transferFrom	v	v	v	v

v: true, x: false

Implemented Action

Fixed in 51b4412. Adjusted the table as per the recommendation.

Secure your system.

Request a service

START NOW →

ABOUT

CONTACT US

SMART CONTRACT AUDITING

PENETRATION TESTING

AUDIT REPORTS

PRIVACY POLICY

TERMS OF SERVICE