SALUS SECURITY

Mar 2024

CODE SECURITY ASSESSMENT

ORA

Overview

Project Summary

- Name: ORA-IMO
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/ora-io/IMOContract-Audit
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	ORA-IMO
Version	v1
Type	Solidity
Dates	Mar 25 2024
Logs	Mar 25 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	2
Total informational issues	4
Total	7

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Not sold IMtoken will lock in contracts	Medium	Business Logic	Pending
2	Users lose the right to purchase MAX_PER_USER tokens	Low	Business Logic	Pending
3	Missing events for functions that change critical state	Low	Logging	Pending
4	Redundant code	Informational	Redundancy	Pending
5	Missing two-step transfer ownership pattern	Informational	Business logic	Pending
6	Use of floating pragma	Informational	Configuration	Pending
7	Missing zero address checks	Informational	Data Validation	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Not sold IMtoken will lock in contracts	
Severity: Medium	Category: Business logic
Target: - src/IMO.sol	

Description

IMO.sol:L151-L158

```
function moveIMOTokens(address to) external onlyOwner onlyBeforeBuyStart {
    require(to != address(0), "Invalid address");

    uint256 imoTokenBalance = imoToken.balanceOf(address(this));
    require(imoTokenBalance > 0, "No IMO tokens to transfer");

    require(imoToken.transfer(to, imoTokenBalance), "Transfer failed");
}
```

Due to the restriction of the onlyBeforeBuyStart modifier, the moveIMOToken function can only be called before the start of the sale period. Therefore, after the end of the sale period, any unsold IMOTokens will remain locked in the contract and cannot be withdrawn, resulting in the loss of IMtoken.

Recommendation

Consider modifying the moveIMOTokens function so that it can transfer IMO tokens after the buyEndTime.

2. Users lose the right to purchase MAX_PER_USER tokens

Severity: Low	Category: Business logic
Target: - src/IMO.sol	

Description

The refund is not always zero. When the refund is not zero, it means that the user spends less than buyAmount during the purchase, but the accounting still uses buyAmount. This would cause userBuySpent[msg.sender] to be larger than the expected value, potentially causing the user to lose the right to purchase MAX_PER_USER tokens.

IMO.sol:L77-L80

```
(uint256 imoAmount, uint256 refund) = buyableIMOAmount(buyAmount);
userBuySpent[msg.sender] = userBuySpent[msg.sender] + buyAmount;
```

Recommendation

Consider the scenario when the refund is not zero. It is recommended to modify it as follows:

```
(uint256 imoAmount, uint256 refund) = buyableIMOAmount(buyAmount);
userBuySpent[msg.sender] = userBuySpent[msg.sender] + buyAmount - refund;
```

3. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - src/IMO.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the IMO contract, events are lacking in the privileged setter functions (e.g. functions setTokenPrice, setBuyPeriod, setBuyTokenAddress and setIMOTokenAddress).

Recommendation

It is recommended to emit events for critical state changes.

2.3 Informational Findings

4. Redundant code

Severity: Informational	Category: Redundancy
Target: - src/IMO.sol	

Description

1. buyAmount > buyAllowed is equivalent to buyAmount > MAX_PER_USER - userBuySpent[msg.sender], which also means buyAmount + userBuySpent[msg.sender] > MAX_PER_USER.
The precondition is `buyAmount + userBuySpent[msg.sender] <= MAX_PER_USER`, so the code block of the if statement will never be executed.

IMO.sol:L120-L141

```
function buyableIMOAmount(uint256 buyAmount) public view returns (uint256 imoAmount, uint256 refund) {
    ...
    require(buyAmount + userBuySpent[msg.sender] <= MAX_PER_USER, "User accumulated buy token amount exceed maximum");
    uint256 buyAllowed = MAX_PER_USER - userBuySpent[msg.sender];

    uint256 buyAmountToSpend = buyAmount;
    if (buyAmount > buyAllowed) {
        buyAmountToSpend = buyAllowed;
        refund = buyAmount - buyAllowed;
    } else {
        refund = 0;
    }
    ...
}
```

2. Unused storage variables in contracts use up storage slots and increase contract size and gas usage at deployment and initialization.

IMO.sol:L24

```
mapping(address => uint256) public stakes;
```

Recommendation

Consider removing the if-else block and keeping only the assignment refund = 0.

5. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - src/IMO.sol	

Description

The IMO contract inherits from the Ownable contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

6. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - src/IMO.sol	

Description

```
pragma solidity ^0.8.24;
```

The IMO uses a floating compiler version ^0.8.24.

Using a floating pragma ^0.8.24 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

7. Missing zero address checks

Severity: Informational	Category: Data Validation
Target: - src/IMO.sol	

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, this precautionary step is absent for address variables buyToken.

IMO.sol:L58, L103

```
buyToken = IERC20(_buyTokenAddress);
```

Recommendation

Consider adding zero address checks for address variables buyToken.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 3a4ebaf:

File	SHA-1 hash
src/IMO.sol	1140af2f956f15710d6e1dc3ba462450f4f5d366