

SALUS SECURITY

DEC 2023



CODE SECURITY ASSESSMENT

W O M

Overview

Project Summary

- Name: wom - WomToken
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See [Appendix - 1](#)

Project Dashboard

Application Summary

Name	wom - WomToken
Version	v1
Type	Solidity
Dates	Dec 18 2023
Logs	Dec 18 2023

Vulnerability Summary

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

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	4
1.1 About SALUS	4
1.2 Audit Breakdown	4
1.3 Disclaimer	4
Findings	5
2.1 Summary of Findings	5
2.2 Notable Findings	6
1. Centralization risk with owner role	6
2. Centralization risk with initial token distribution	7
2.3 Informational Findings	8
3. Use of floating pragma	8
4. Use of the magic number	9
Appendix	10
Appendix 1 - Files in Scope	10

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	Centralization risk with owner role	Low	Centralization	Pending
2	Centralization risk with initial token distribution	Low	Centralization	Pending
3	Use of floating pragma	Informational	Configuration	Pending
4	Use of the magic number	Informational	Code Quality	Pending

2.2 Notable Findings

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

1. Centralization risk with owner role	
Severity: Low	Category: Centralization
Target: <ul style="list-style-type: none">- erc20/womerc20.sol	

Description

There is a privileged owner role in the WomToken contract. The owner can control whether tokens can be transferred.

Should the owner's private key be compromised, an attacker can invoke the pause() function and renounce ownership. Thus, tokens can not be transferred anymore.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

2. Centralization risk with initial token distribution

Severity: Low

Category: Centralization

Target:

- erc20/womerc20.sol

Description

When the contract is deployed, \$WOM is sent to one account. This account then has full control over the token distribution. If it is an EOA account, any compromise of its private key could drastically affect the project – for example, attackers could dump the price of \$WOM on the DEX if they gain access to the private key.

Recommendation

It is recommended to transfer tokens to a multi-sig account and promote transparency by providing a breakdown of the intended initial token distribution in a public location.

2.3 Informational Findings

3. Use of floating pragma

Severity: Informational

Category: Configuration

Target:

- erc20/womerc20.sol

Description

```
pragma solidity ^0.8.20;
```

The contract WomToken uses a floating compiler version ^0.8.20.

Using a floating pragma ^0.8.20 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.

4. Use of the magic number

Severity: Informational

Category: Code Quality

Target:

- erc20/womerc20.sol

Description

There are a few occurrences of literal values being used. To improve the code's readability and facilitate refactoring, consider defining a constant for every magic number, giving it a clear and self-explanatory name.

erc20/womerc20.sol:L891

```
_mint(msg.sender, 400000000000000000000000000 * 10 ** decimals());
```

Recommendation

Consider defining a constant variable INITIAL_SUPPLY for the magic number 4000000000000000000000000000.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file provided by the client:

File	SHA-1 hash
erc20/womerc20.sol	1958be78cf86290b369fb9c1c00c498c528fdddd