SALUS SECURITY

NOV 2023

CODE SECURITY ASSESSMENT

MAILZERO

Overview

Project Summary

- Name: MailZero - MailZero Token
- Platform: Ethereum
- Language: Solidity
- Repository:
    - https://github.com/MailZeroProtocol/mailzero-token
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	MailZero - MailZero Token
Version	v2
Type	Solidity
Dates	Nov 29 2023
Logs	Nov 28 2023; Nov 29 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	3
Total	5

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	Centralization risk	Low	Centralization	Acknowledged
2	Using an unstable version of Openzeppelin contracts	Low	Configuration	Acknowledged
3	Use of deprecated ERC20Permit draft contract	Informational	Code Quality	Acknowledged
4	Use of the magic number	Informational	Code Quality	Acknowledged
5	Avoid using forge update	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Centralization risk

Severity: Low	Category: Centralization
Target: - src/OurToken.sol	

Description

The ZeroToken contract has a privileged owner role. The owner of the ZeroToken contract can mint a specified number of tokens for a specified user.

src/OurToken.sol:L15-L17

```
function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
}
```

If the privileged account is a plain EOA account, this can be worrisome and pose a risk to the other users. For example, if the owner's private key is compromised, an attacker can mint any number of tokens for anyone.

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.

Status

This issue has been acknowledged by the team.

2. Using an unstable version of Openzeppelin contracts

Severity: Low	Category: Configuration

Target:
- src/OurToken.sol

Description

A parameter is passed to the Ownable contract during the deployment, which means that the version of Openzeppelin contracts should be at least v5.0.0.

src/OurToken.sol:L10

```
constructor() ERC20("MailZero Token", "MAIX") Ownable(msg.sender) ERC20Permit("MailZero Token")
```

However, an ERC20Permit draft contract is also used, which has been deprecated since v4.9.0 and has been removed in v5.0.0. Thus, the version used is unstable, which may include unexpected breaking changes between releases.

src/OurToken.sol:L7

```
import {ERC20Permit} from "@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol";
```

Recommendation

Consider pinning imported dependencies to a stable version and fixing the dependencies.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Use of deprecated ERC20Permit draft contract

Severity: Informational	Category: Code Quality
Target: - src/OurToken.sol	

Description

The ZeroToken contract imports draft-ERC20Permit.sol, a deprecated OpenZeppelin draft contract.

src/OurToken.sol:L7

```
import {ERC20Permit} from
"@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol";
```

Recommendation

Since EIP-2612 is no longer a draft, consider utilizing ERC20Permit.sol instead.

Status

This issue has been acknowledged by the team.

4. Use of the magic number

Severity: Informational	Category: Code Quality
Target: - src/OurToken.sol	

Description

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.

src/OurToken.sol:L10-L12

```
constructor() ERC20("MailZero Token", "MAIX") Ownable(msg.sender) ERC20Permit("MailZero
Token") {
        _mint(msg.sender, 1000000000 * 10 ** decimals());
}
```

Recommendation

Consider defining a constant variable INITIAL_SUPPLY for the magic number 1000000000.

Status

This issue has been acknowledged by the team.

5. Avoid using forge update

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

Description

When installing via git, it is recommended to use tagged releases instead of the master branch, which is a development branch. The release process involves security measures that the master branch does not guarantee.

Foundry installs the latest version initially, but subsequent `forge update` commands will use the master branch.

Recommendation

Consider not using `forge update` to avoid installing undesired versions that may break functionality.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file in commit a740c60:

File	SHA-1 hash
src/OurToken.sol	32cd4d426fdafd9e3e7e361ee8fb6e5953a3c590