SALUS SECURITY

APR 2025

CODE SECURITY ASSESSMENT

EPT TOKEN

Overview

Project Summary

- Name: EPT - Token
- Platform: The Ethereum Blockchain
- Language: Solidity
- Repository:
    - 0x7a8B288294Cb9EA4537e1D8A003Fb38651b9393f
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	EPT- Token
Version	v3
Type	Solidity
Dates	Apr 17 2025
Logs	Mar 05 2025; Mar 05 2025; Apr 17 2025

Vulnerability Summary

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

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. Wrong cap configuration prevents future token minting 6
2.3 Informational Findings 8
2. Missing two-step transfer ownership pattern 8
Appendix **9**
Appendix 1 - Files in Scope 9

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	Wrong cap configuration prevents future token minting	Low	Business logic	Mitigated
2	Missing two-step transfer ownership pattern	Informational	Business logic	Resolved

2.2 Notable Findings

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

1. Wrong cap configuration prevents future token minting

Severity: Low	Category: Business logic

Target:
- contracts/ERC20Token/EPTToken.sol

Description

In the `EPTToken` contract, the `initialOwner` minted `_initialSupply*10**18` number of tokens during the constructor process. Then, the `totalSupply()` is equal to `_initialSupply*10**18`, and also equal to `cap()`.

contracts/ERC20Token/EPTToken.sol:L10-L17

```
constructor(...) ERC20(_name, _symbol) ERC20Capped(_initialSupply*10**18)
Ownable(initialOwner){
    ERC20._mint(initialOwner, _initialSupply*10**18);
}
```

According to the `ERC20` and `ERC20Capped` contracts code, the contract can't mint tokens anymore unless the token is burned. But the `EPTToken` contract has no burn function. Therefore, the mint mechanism fails.

contracts/ERC20Token/EPTToken.sol:L19-L21

```
function mint(address _to, uint256 _amount) onlyOwner external {
    _mint(_to, _amount);
}
```

@openzeppelin/contracts/token/ERC20/ERC20.sol:L221-L225

```
function _mint(address account, uint256 value) internal {
    if (account == address(0)) {
        revert ERC20InvalidReceiver(address(0));
    }
    _update(address(0), account, value);
}
```

@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol:L45-L55

```
function _update(address from, address to, uint256 value) internal virtual override {
    super._update(from, to, value);

    if (from == address(0)) {
        uint256 maxSupply = cap();
        uint256 supply = totalSupply();
        if (supply > maxSupply) {
            revert ERC20ExceededCap(supply, maxSupply);
        }
    }
}
```

Recommendation

Refactor the mint mechanism or remove the redundant code.

Status

This issue has been mitigated by the team. The team has stated that there are no plans for additional token issuance in the future.

2.3 Informational Findings

2. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - contracts/ERC20Token/EPTToken.sol	

Description

The `ProjectLib` contract uses the `transferProjectOwnership` function which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern. This simpler mechanism can be useful for quick tests, but projects with production concerns are likely to outgrow it. Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership, e.g. mistakenly.

Recommendation

It is recommended to implement a two-step transfer of ownership mechanism where the ownership is transferred and later claimed by a new owner to confirm the whole process and prevent lockout.

Status

The team has resolved this issue by renouncing ownership, which means the team no longer requires owner privileges.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file from address 0x7a8B288294Cb9EA4537e1D8A003Fb38651b9393f:

File	SHA-1 hash
contracts/ERC20Token/EPTToken.sol	11e386557b2643bdfacf21f8afc4032a3c79afd8