SALUS SECURITY

JULY 2023

CODE SECURITY ASSESSMENT

ISE

Overview

Project Summary

- Name: ISE - EBU Token
- Address: 0x2b85cF630fccE4a97843b8930833DBd086d99Be9
- Platform: BNB Smart Chain
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	ISE - EBU Token
Version	v1
Type	Solidity
Dates	July 28 2023
Logs	July 28 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	3
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	Uninitialized state variable	Low	Business Logic	Pending
2	Unexpected token burn via _transfer()	Low	Business Logic	Pending
3	Centralization risk with initial token distribution	Low	Centralization	Pending
4	Use of floating pragma	Informational	Configuration	Pending
5	SafeMath is not required since Solidity 0.8.0	Informational	Redundancy	Pending
6	Unclear state variable name	Informational	Configuration	Pending
7	Missing event for critical parameters change	Informational	Logging	Pending

2.2 Notable Findings

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

1. Uninitialized state variable

Severity: Low	Category: Business Logic
Target: - EBU.sol	

Description

EBU.sol:L289

```
uint256 public minTotalSupply;
```

The state variable minTotalSupply is never initialized, which is used to determine whether to burn a certain amount of token. The default value of minTotalSupply is 0 if not initialized.

EBU.sol:L442-L463

```
function _burn(
        address account,
        uint256 amount
) internal virtual returns (bool) {
        require(account != address(0), "ERC20: burn from the zero address");
        if (_totalCirculation > minTotalSupply + amount) {
            _beforeTokenTransfer(account, address(0), amount);
            uint256 accountBalance = _balances[account];
            require(
                accountBalance >= amount,
                "ERC20: burn amount exceeds balance"
            );
            unchecked {
                _balances[account] = accountBalance - amount;
                _balances[address(0)] += amount;
            }
            emit Transfer(account, address(0), amount);
            _afterTokenTransfer(account, address(0), amount);
            return true;
        }
        return false;
}
```

Recommendation

Consider adding a privileged function to update minTotalSupply or initialize it in the constructor.

2. Unexpected token burn via _transfer()

Severity: Low	Category: Business Logic
Target: - EBU.sol	

Description

The _transfer function checks if `from` and `recipient` addresses are not address(0). However, if a user transfers tokens to address(1), the target address will then be replaced with address(0). This kind of transfer is similar to burning tokens, except without the pre-burning verification.

EBU.sol:L411-L432

```
function _transfer(
        address from,
        address recipient,
        uint256 amount
) internal virtual {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");
        address to = recipient;
        if (address(1) == recipient) to = address(0);
        _beforeTokenTransfer(from, to, amount);
        uint256 fromBalance = _balances[from];
        require(
            fromBalance >= amount,
            "ERC20: transfer amount exceeds balance"
        );
        unchecked {
            _balances[from] = fromBalance - amount;
        }
        _balances[to] += amount;
        emit Transfer(from, to, amount);
        _afterTokenTransfer(from, to, amount);
}
```

Recommendation

It is recommended to remove the assignment logic when the recipient address is address(1) in transfer(). Additionally, to support burning tokens on behalf of other users, it would be beneficial to include a burnFrom() function instead of using transferFrom().

3. Centralization risk with initial token distribution	
Severity: Low	Category: Centralization
Target: - EBU.sol	

Description

When the contract is deployed, $EBU is sent to 7 EOA accounts. These accounts then have full control over the token distribution, so any compromise of their private keys could have a drastic effect on the project – for example, attackers could dump the price of $EBU on the DEX if they gain access to the private key.

Recommendation

It is recommended to promote transparency by providing a breakdown of the intended initial token distribution in a public location.

2.3 Informational Findings

4. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - EBU.sol	

Description

```
pragma solidity ^0.8.1;
```

The EBU contract uses a floating compiler version ^0.8.1.

Using a floating pragma ^0.8.1 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.

5. SafeMath is not required since Solidity 0.8.0

Severity: Informational	Category: Redundancy
Target: - EBU.sol	

Description

The EBU contract uses SafeMath library for mathematical operations. SafeMath is generally used to avoid overflow and underflow. But there are inbuilt overflow and underflow operations from solidity version 0.8.0. So, the use of SafeMath library is redundant.

Recommendation

Consider removing the SafeMathUpgradeable library.

6. Unclear state variable name

Severity: Informational	Category: Configuration
Target: - EBU.sol	

Description

The state variable `m` represents the address of the current manager. For clarity and readability, it is recommended to rename it to a more meaningful name, for example, `manager`.

EBU.sol:L290

```
address public m;
```

Recommendation

Consider renaming the state variable `m`.

7. Missing event for critical parameters change

Severity: Informational	Category: Logging
Target: - EBU.sol	

Description

Important parameter or configuration changes should trigger an event to enable tracking off-chain, but the setManager() function does not emit an event.

EBU.sol:L296-L300

```
function setManager(address account) public {
        if (owner() == _msgSender() || m == _msgSender()) {
            m = account;
        }
}
```

Recommendation

It is recommended to emit a ManagerChanged event after updating the address of manager.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file from address 0x2b85cF630fccE4a97843b8930833DBd086d99Be9:

File	SHA-1 hash
EBU.sol	2296b846f70c9a66c180fa7a04a3bb39b781187e