SALUS SECURITY

MAY 2025

CODE SECURITY ASSESSMENT

CUDIS

Overview

Project Summary

- Name:CUDIS
- Platform: The BSC Blockchain
- Language: Solidity
- Address:
    - https://bscscan.com/token/0xC66B64E17215BA16C692f44383F173170B3C5343
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	CUDIS
Version	v2
Type	Solidity
Dates	May 13 2025
Logs	Apr 27 2025; May 13 2025

Vulnerability Summary

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

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	Informational	Centralization	Resolved
2	Can not burn tokens when transfersenabled is false	Informational	Business Logic	Resolved
3	Multiple access-control scheme is used	Informational	Configuration	Acknowledged

2.2 Informational Findings

1. Centralization risk

Severity: Low	Category: Centralization
Target: - contracts/CUDIS.sol	

Description

The `CUDIS` contract has privileged accounts. These privileged accounts can mint tokens without capped(even if the file imports the `ERC20Capped` contract) by using the `mint()` functions and transfer token without enabling the `transfersEnabled`.

If privileged accounts' private key or admin's is compromised, an attacker can mint and transfer the token easily.

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.

Status

This issue has been resolved by the team. The team has moved the privileged address to muti-sign wallet.

2. Can not burn tokens when transfersenabled is false

Severity: Informational	Category: Business Logic
Target: - contracts/CUDIS.sol	

Description

The CUDIS contract inherits the ERC20Burnable contract, so users can call the `burn()` and `burnFrom()` functions, but when `transfersenabled` is false, users can't burn their tokens.

contracts/CUDIS.sol:L10

```
contract CUDIS is ERC20, ERC20Burnable, Ownable, AccessControlEnumerable {
```

ERC20Burnable

```
function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }
function burnFrom(address account, uint256 amount) public virtual {
    _spendAllowance(account, _msgSender(), amount);
    _burn(account, amount);
}
```

Recommendation

Consider adding a check in `_beforeTokenTransfer()` function, like:

```
-   require(from == address(0) || from == owner() || hasRole(TRANSFER_ROLE, from),
"ERC20: transfers not enabled");
+   require(to == address(0) || from == address(0) || from == owner() ||
hasRole(TRANSFER_ROLE, from), "ERC20: transfers not enabled");
```

Status

This issue has been resolved by the team.

3. Multiple access-control scheme is used

Severity: Informational	Category: Configuration
Target: - contracts/CUDIS.sol	

Description

The contract employs both OpenZeppelin's Ownable and AccessControlEnumerable for permission management. While `transferOwnership()` reassigns the owner, it does not automatically grant the `DEFAULT_ADMIN_ROLE` to the new owner nor revoke it from the previous owner. As a result, the new owner loses the ability to manage `MINTER_ROLE` and `TRANSFER_ROLE` assignments, while the former owner (no longer the contract owner) retains role-administration privileges.

Recommendation

Consolidate to a single permission scheme (either Ownable or AccessControl) to avoid split administration.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered on-chain token:

0xC66B64E17215BA16C692f44383F173170B3C5343