CredShields

Smart Contract Audit

June 23, 2026 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Cluster Protocol between June 22, 2026, and June 23, 2026. A retest was performed on June 23, 2026.

Author

Shashank (Co-founder, CredShields) shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead), Shreyas Koli (Auditor), Prasad Kuri (Auditor), Neel Shah (Auditor)

Prepared for

`Cluster Protocol`

Table of Contents

Table of Contents **2**

1. Executive Summary ——————————- **3**

State of Security 4

2. The Methodology ——————————- **5**

2.1 Preparation Phase 5

2.1.1 Scope 5

2.1.2 Documentation 5

2.1.3 Audit Goals 6

2.2 Retesting Phase 6

2.3 Vulnerability classification and severity 6

2.4 CredShields staff 8

3. Findings Summary ——————————- **9**

3.1 Findings Overview 9

3.1.1 Vulnerability Summary 9

5. Bug Reports —————————— **10**

Bug ID #M001 [Acknowledged] 10

MAX_SUPPLY Is a Per-Chain Cap, Not a Global Cap 10

Bug ID #L001 [Acknowledged] 11

Constructor Silently Skips Initial Mint on Misconfiguration 11

Bug ID #L002 [Acknowledged] 12

Outdated Pragma 12

Bug ID #I001 [Acknowledged] 13

setCCIPAdmin Allows Setting the Admin to the Current Value 13

6. The Disclosure ——————————- **14**

1. Executive Summary ----------

Cluster Protocol engaged CredShields to perform a smart contract audit from June 22, 2026, to June 23, 2026. During this timeframe, 4 vulnerabilities were identified. **A retest was performed on June 23, 2026, and all the bugs have been addressed.**

During the audit, 0 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Cluster Protocol" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
CP Token Smart Contract	0	0	1	2	1	0	**4**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in CP Token Smart Contract's scope during the testing window while abiding by the policies set forth by Cluster Protocol's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both Cluster Protocol's internal security and development teams to not only identify specific vulnerabilities but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at Cluster Protocol can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Cluster Protocol can future-proof its security posture and protect its assets.

2. The Methodology ----------

Cluster Protocol engaged CredShields to perform a CP Token Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation Phase

The CredShields team meticulously reviewed all provided documents and comments in the smart contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from June 22, 2026, to June 23, 2026, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed upon:

IN SCOPE ASSETS
Audited Scope: https://github.com/clusterprotocol/CPTOKEN/tree/62cfed0a19ac056518dd78f063b7a3106c08368d

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields employs a combination of in-house tools and thorough manual review processes to deliver comprehensive smart contract security audits. The majority of the audit involves manual inspection of the contract's source code, guided by OWASP's Smart Contract Security Weakness Enumeration (SCWE) framework and an extended, self-developed checklist built from industry best practices. The team focuses on deeply understanding the contract's core logic, designing targeted test cases, and assessing business logic for potential vulnerabilities across OWASP's identified weakness classes.

CredShields aligns its auditing methodology with the OWASP Smart Contract Security projects, including the Smart Contract Security Verification Standard (SCSVS), the Smart Contract Weakness Enumeration (SCWE), and the Smart Contract Secure Testing Guide (SCSTG). These frameworks, actively contributed to and co-developed by the CredShields team, aim to bring consistency, clarity, and depth to smart contract security assessments. By adhering to these OWASP standards, we ensure that each audit is performed against a transparent, community-driven, and technically robust baseline. This approach enables us to deliver structured, high-quality audits that address both common and complex smart contract vulnerabilities systematically.

2.2 Retesting Phase

Cluster Protocol is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat

agents, Vulnerability factors, and Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	None	Low	Medium
		LOW	MEDIUM	HIGH
Likelihood				

Overall, the categories can be defined as described below -

1. **Informational**

   We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. **Low**

   Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. **Medium**

   Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities

can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- Shashank, Co-founder CredShields shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have about the engagement or this document.

3. Findings Summary ----------

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by asset and OWASP SCWE classification. Each asset section includes a summary highlighting the key risks and observations. The table in the executive summary presents the total number of identified security vulnerabilities per asset, categorized by risk severity based on the OWASP Smart Contract Security Weakness Enumeration framework.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 4 security vulnerabilities were identified in the asset.

Vulnerability Title	Severity	Vulnerability Type	Status
MAX_SUPPLY Is a Per-Chain Cap, Not a Global Cap	Medium	Business Logic (SC03-LogicErrors)	Acknowledged
Constructor Silently Skips Initial Mint on Misconfiguration	Low	Lack of Input Validation (SC04-Lack Of Input Validation)	Acknowledged
Outdated Pragma	Low	Outdated Compiler Version (SCWE-061)	Acknowledged
setCCIPAdmin Allows Setting the Admin to the Current Value	Informational	Lack of Input Validation (SC04-Lack Of Input Validation)	Acknowledged

Table: Findings and Remediations

5. Bug Reports ----------

Bug ID #M001 [Acknowledged]

MAX_SUPPLY Is a Per-Chain Cap, Not a Global Cap

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
CPToken is deployed at the same address on three chains (Base, BNB Chain, Mantle) and the README markets a "Fixed max supply: 5,000,000,000 CP." In the code, MAX_SUPPLY is a compile-time constant equal to 5_000_000_000 * 1e18, and the only place it is enforced is the mint() function, which compares totalSupply() + amount against MAX_SUPPLY. The crucial detail is that totalSupply() is OpenZeppelin's ERC-20 local accounting variable, it reflects only the supply minted on the chain where the call executes, and it has no awareness of how many tokens exist on the other deployments. The contract holds no global supply state, queries no bridge or registry for aggregate issuance, and exposes no cross-chain accounting hook of any kind.

Affected Code

- CPToken.sol#L111

Impacts
The "fixed 5,000,000,000 CP supply" guarantee marketed to holders is not enforceable on-chain at the protocol level.

Remediation
If a per-chain cap is intended, update the README about a global "Fixed max supply: 5,000,000,000 CP", otherwise change the code to enforce a true global cap.

Retest
Client Comment: Per-chain MAX_SUPPLY is intentional for Chainlink CCT BurnMintTokenPool. Global 5B supply is preserved by burn-on-source / mint-on-destination via CCIP. We will update public docs to reflect this.

Bug ID #L001 [Acknowledged]

Constructor Silently Skips Initial Mint on Misconfiguration

Vulnerability Type
Lack of Input Validation (SC04-Lack Of Input Validation)

Severity
Low

Description
The constructor is the only code path that can mint CPToken's genesis supply to a treasury without holding MINTER_ROLE, so it is effectively a one-shot, unrepeatable operation for the home-chain deployment. Its input validation checks that _admin is non-zero and that _initialSupply does not exceed MAX_SUPPLY, but it never checks that _initialSupply and _treasury are mutually consistent. The actual mint is guarded by the compound condition _initialSupply > 0 && _treasury != address(0). This condition is correct for the intended remote-chain case, where the deployer deliberately passes _initialSupply = 0 and _treasury = address(0). The problem is the asymmetric mistake: if a home-chain deployer passes a real _initialSupply (for example MAX_SUPPLY) but accidentally leaves _treasury as address(0), the guard short-circuits to false, the _mint call is skipped, and the constructor completes successfully. There is no revert, no event, and no other signal that the genesis mint did not happen.

Affected Code

- CPToken.sol#L85-L87

Impacts
A zero treasury address on the home-chain deployment produces a token that deploys cleanly but holds no supply, with no on-chain indication of the failure.

Remediation
If the silent-skip is intended for the remote-chain case, leave it but require coherence so the contradictory case reverts, if (_initialSupply > 0) { require(_treasury != address(0), "treasury required for initial supply"); _mint(_treasury, _initialSupply); }; otherwise no change is needed for correctly-parameterized deploys.

Retest
Client Comment: Valid deploy-time check. Home chain deployed with correct treasury; no impact on live deployment. Will add require in next deployment script/version.

Bug ID #L002 [Acknowledged]

Outdated Pragma

Vulnerability Type
Outdated Compiler Version (SCWE-061)

Severity
Low

Description
The smart contract is using an outdated version of the Solidity compiler specified by the pragma directive i.e. 0.8.24. Solidity is actively developed, and new versions frequently include important security patches, bug fixes, and performance improvements. Using an outdated version exposes the contract to known vulnerabilities that have been addressed in later releases. Additionally, newer versions of Solidity often introduce new language features and optimizations that improve the overall security and efficiency of smart contracts.

Affected Code

- CPToken.sol#L2

Impacts
The use of an outdated Solidity compiler version can have significant negative impacts. Security vulnerabilities that have been identified and patched in newer versions remain exploitable in the deployed contract.
Furthermore, missing out on performance improvements and new language features can result in inefficient code execution and higher gas costs.

Remediation
It is suggested to use the 0.8.34 pragma version.
Reference: https://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-061/

Retest
Client Comment: Noted. 0.8.24 used for deterministic deployment across chains. Will evaluate 0.8.34 for future deployments.

Bug ID #I001 [Acknowledged]

setCCIPAdmin Allows Setting the Admin to the Current Value

Vulnerability Type
Lack of Input Validation (SC04-Lack Of Input Validation)

Severity
Informational

Description
setCCIPAdmin updates the _ccipAdmin address that Chainlink's TokenAdminRegistry trusts for pool registration. It validates that the new admin is not the zero address but does not validate that the new admin differs from the value currently stored in _ccipAdmin. As a result, an admin can call setCCIPAdmin(currentAdmin) and the function will execute fully: it caches the previous value, re-assigns _ccipAdmin to the same address, and emits CCIPAdminTransferred(previous, newAdmin) with previous == newAdmin. On-chain, the state is unchanged, so the write is pure overhead; off-chain, any system that treats CCIPAdminTransferred as a signal that the CCIP admin actually rotated will record a transfer that never happened.

Affected Code

- CPToken.sol#L96-L101

Impacts
Calling setCCIPAdmin with the already-current admin performs a redundant storage write and emits a CCIPAdminTransferred(previous, newAdmin) event in which previous equals newAdmin.

Remediation
It is suggested to add a redundant-value guard so the call reverts when nothing changes.

Retest
Client Comment: Will add redundant admin guard in future version. No security impact on current deployment.

6. The Disclosure ----------

The Reports provided by CredShields are not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.

Your Secure Future Starts Here

CRED SHIELDS

At CredShields, we're more than just auditors. We're your strategic partner in ensuring a secure Web3 future. Our commitment to your success extends beyond the report, offering ongoing support and guidance to protect your digital assets

AICPA SOC
aicpa.org/soc4so

SCS Advocates
OWASP
OWASP Smart Contract Top 10
Security Risks & Vulnerabilities

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com