Audited by

CredShields

Smart Contract Audit

December 05, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Octos between November 25, 2025, and November 25, 2025. A retest was performed on December 04, 2025.

Author

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

Reviewers

Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor), Prasad Kuri (Auditor), Neel Shah (Auditor)

Prepared for

Octos

Table of Contents

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

Octos engaged CredShields to perform a smart contract audit from November 25, 2025, to November 25, 2025. During this timeframe, 7 vulnerabilities were identified. A retest was performed on December 04, 2025, and all the bugs have been addressed.

During the audit, 1 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Octos" and should be prioritized for remediation.

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
Smart Contract	0	1	1	2	1	2	7

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Smart Contract's scope during the testing window while abiding by the policies set forth by Octos'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 Octos'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 Octos can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Octos can future-proof its security posture and protect its assets.

2. The Methodology ———————————

Octos engaged CredShields to perform a 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 November 25, 2025, to November 25, 2025, 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 Commit: https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1

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

Octos 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, 7 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
H001 — User Can Stake Locked Tokens And Claim Rewards	High	Business Logic (SC03-LogicErrors)
M001 — User Receive Rewards Based on Updated Interest Rate	Medium	Business Logic (SC03-LogicErrors)
L001 — Owner Can Set Arbitrary Interest Rate Beyond Intended Bounds	Low	Missing Input Validation (SC04-Lack Of Input Validation)
L001 — Missing events in important functions	Low	Missing Best Practices
I001 — Use Ownable2Step	Informational	Missing Best Practices
G001 — Gas Optimization in Require/Revert Statements	Gas	SCWE-082 Gas Optimization
G002 — Gas Optimization in Increments	Gas	SCWE-082 Gas Optimization

Table: Findings in Smart Contracts

4. Remediation Status ————————

Octos is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. A retest was performed on December 04, 2025, and all the issues have been addressed.

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	**REMEDIATION STATUS**
H001 — User Can Stake Locked Tokens And Claim Rewards	High	Fixed [2025-12-04]
M001 — User Receive Rewards Based on Updated Interest Rate	Medium	Fixed [2025-12-04]
L001 — Owner Can Set Arbitrary Interest Rate Beyond Intended Bounds	Low	Fixed [2025-12-04]
L001 — Missing events in important functions	Low	Fixed [2025-12-04]
I001 — Use Ownable2Step	Informational	Fixed [2025-12-04]
G001 — Gas Optimization in Require/Revert Statements	Gas	Partially Fixed [2025-12-04]
G002 — Gas Optimization in Increments	Gas	Fixed [2025-12-04]

Table: Summary of findings and status of remediation

5. Bug Reports ————————————

Bug ID #H001 [Fixed]

User Can Stake Locked Tokens And Claim Rewards

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
High

Description
The contract implements a token lock mechanism via lockedTokens(...) and enforces it by overriding transfer and transferFrom, which call getLockedAmount to prevent movement of locked balances. The stake(...) function allows a user to move tokens into the staking contract by calling _transfer(msg.sender, address(this), amount). This function transfers tokens without invoking the public transfer/transferFrom logic that enforces locks. The root cause is the use of the internal _transfer call within stake, which bypasses the overridden transfer checks and thus violates the intended locking invariant.

Affected Code

- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L815

Impacts
A user whose balance was locked by the owner can nevertheless call stake(...) to move those locked tokens into the staking contract; after the staking period the user will receive the original tokens back plus minted rewards.

Remediation
Require the same locked-balance check in stake before calling _transfer, ensuring internal transfers respect the lock invariant.

Retest
This issue has been fixed by calling transfer function in stake instead of _transfer.

Bug ID #M001 [Fixed]

User Receive Rewards Based on Updated Interest Rate

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
The staking workflow records amount, startTime, and duration for each stake in StakeInfo, and users later call claimStake to receive their principal plus rewards. The reward calculation reads the global interestRatePerYearBP, which the owner can update at any time through setInterestRate. The value of interestRatePerYearBP at the time of claiming—not at the time of staking—is used in the reward formula. The root cause is that StakeInfo does not store the interest rate effective at stake creation, and claimStake references the mutable global rate.

Affected Code

- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L828

Impacts
Interest payouts can be arbitrarily altered after users have already staked, breaking predictable and fair yield assumptions.

Remediation
Record the active interest rate at stake creation and use the stored value in reward calculations.

Retest
This issue has been fixed by recording the active interest rate at stake creation and use the stored value in reward calculations.

Bug ID #L001 [Fixed]

Owner Can Set Arbitrary Interest Rate Beyond Intended Bounds

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

Severity
Low

Description
The staking system issues rewards based on interestRatePerYearBP, which is configured through the owner-only setInterestRate function. The intended workflow is that the owner sets an annual percentage yield using a string input, which is parsed into basis points by parseInterestRate. The reward formula in claimStake relies on this value to compute yearly interest. However, setInterestRate accepts any parsed value with no minimum or maximum boundary checks. The root cause is the absence of constraints in both setInterestRate and parseInterestRate, permitting the owner to set excessively large interest values unintentionally.

Affected Code

- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L870

Impacts
The interest rate can be set extremely large values, leading to disproportionate reward minting during claimStake.

Remediation
Introduce explicit minimum and maximum allowable interest rates in setInterestRate to ensure reward calculations remain economically bounded.

Retest
This issue has been fixed by checking minimum and maximum allowable interest rates in setInterestRate.

Bug ID #L002 [Fixed]

Missing events in important functions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.

The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

Affected Code
- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L866
- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L874
- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L892
- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L896

Impacts
Events are used to track the transactions off-chain and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Remediation
Consider emitting events for important functions to keep track of them.

Retest
This issue has been fixed by emitting events for important functions.

Bug ID #I001 [Fixed]

Use Ownable2Step

Vulnerability Type
Missing Best Practices

Severity
Informational

Description
The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

Affected Code

- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L724

Impacts
Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

Remediation
It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Retest:
This issue has been fixed by using Ownable2Step.

Bug ID #G001 [Partially Fixed]

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
The **require/revert** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.
Affected Code

- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L810
- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L823
- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L817

Impacts
Having longer require/revert strings than **32 bytes** cost a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside **require/revert** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest
This issue has been partially fixed.

Bug ID#G002 [Fixed]

Gas Optimization in Increments

Vulnerability Type
Gas optimization (SCWE-082)

Severity
Gas

Description
The contract uses two for loops, which use post increments for the variable "**i**".
The contract can save some gas by changing this to **++i**.
++i costs less gas compared to **i++** or **i += 1** for unsigned integers. In **i++**, the compiler has to create a temporary variable to store the initial value. This is not the case with **++i** in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Vulnerable Code

- https://bscscan.com/address/0x84b9c39affba189110f9ad6c2318b127508cf5f1#code#L919

Impacts
Using **i++** instead of **++i** costs the contract deployment around 600 more gas units.

Remediation
It is recommended to switch to **++i** and change the code accordingly so the function logic remains the same and meanwhile saves some gas.

Retest
This issue has been fixed.

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

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

Audited by

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com