Audited by CRED SHIELDS

CredShields

Smart Contract Audit

November 3, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Capx AI between October 24th, 2025, and October 30th, 2025. A retest was performed on October 30th, 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

Capx AI

Table of Contents

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

Capx AI engaged CredShields to perform a smart contract audit from October 24th, 2025, to October 30th, 2025. During this timeframe, 10 vulnerabilities were identified. **A retest was performed on October 30th, 2025, 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 "Capx AI" and should be prioritized for remediation; 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
Capx Launchpad	0	0	0	4	0	6	**10**
	0	**0**	**0**	**4**	**0**	**6**	**10**

Table: Vulnerabilities Per Asset in Scope

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

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

Capx AI engaged CredShields to perform a Capx Launchpad 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 October 24th, 2025, to October 30th, 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
• https://github.com/Capx-AI/Capx-Launchpad-Contracts/tree/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95 • https://github.com/Capx-AI/Capx-Launchpad-Contracts/tree/e6a0b0e049b673543a682f0019862aa00fa96395

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

Capx AI 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, 10 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
Missing _gap In Upgradable Contract	Low	Storage Layout Conflict
Missing events in important functions	Low	Missing Best Practices
Floating and Outdated Pragma	Low	Floating Pragma (SCWE-060)
Use Ownable2Step	Low	Missing Best Practices
Cheaper Inequalities in require()	Gas	Gas Optimization (SCWE-082)
Cheaper conditional operators	Gas	Gas Optimization (SCWE-082)
Gas Optimization in Require/Revert Statements	Gas	Gas Optimization (SCWE-082)

Splitting Require/Revert Statements	Gas	Gas Optimization (SCWE-082)
Gas Optimization in Increments	Gas	Gas Optimization (SCWE-082)
Cheaper Inequalities in if()	Gas	Gas Optimization (SCWE-082)

Table: Findings in Smart Contracts

4. Remediation Status ----------

Capx AI is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on October 30th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Missing _gap In Upgradable Contract	Low	Fixed [Oct 30, 2025]
Missing events in important functions	Low	Fixed [Oct 30, 2025]
Floating and Outdated Pragma	Low	Partially Fixed [Oct 30, 2025]
Use Ownable2Step	Low	Fixed [Oct 30, 2025]
Cheaper Inequalities in require()	Gas	Won't Fix [Oct 30, 2025]
Cheaper conditional operators	Gas	Won't Fix [Oct 30, 2025]
Gas Optimization in Require/Revert Statements	Gas	Fixed [Oct 30, 2025]
Splitting Require/Revert Statements	Gas	Won't Fix [Oct 30, 2025]
Gas Optimization in Increments	Gas	Fixed [Oct 30, 2025]
Cheaper Inequalities in if()	Gas	Won't Fix [Oct 30, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #L001 [Fixed]

Missing _gap In Upgradable Contract

Vulnerability Type
Storage Layout Conflict

Severity
Low

Description
The contract is UUPS upgradeable but does not include reserved storage gaps (e.g., uint256[50] private __gap;) to safely add new variables in future versions without risking storage collisions across upgrades. While not an immediate exploit, this increases the risk of state corruption during future upgrades.

Affected Code
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/develop/contracts/PreSale.sol

Impacts
Future upgrades may unintentionally overwrite existing storage slots, leading to state corruption, loss of funds, or broken access control after an upgrade.

Remediation
Add storage gaps in this contract and any future upgradeable contracts/libraries. Follow OpenZeppelin's storage gap pattern (e.g., uint256[50] private __gap;) and carefully manage layout across versions.

Retest
This issue has been fixed in b0ece48

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://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L937-L940

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 in b0ece48

Bug ID #L003 [Partially Fixed]

Floating and Outdated Pragma

Vulnerability Type
Floating Pragma (SCWE-060)

Severity
Low

Description
Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.
The contract allowed floating or unlocked pragma to be used, i.e., ^0.8.27. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

Affected Code

- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L2
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/develop/contracts/CapxRewardDistributor.sol#L2

Impacts
If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.
Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is low.

Remediation
Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.29 pragma version
Reference: https://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-060/

Retest
This issue has been partially fixed.

Client's comment: We locked to 0.8.27 (current version) rather than upgrading to 0.8.29 because:

Minimize testing surface: Contracts already tested with 0.8.27
Real issue was floating pragma: Both 0.8.27 and 0.8.29 are recent and secure
Deliberate upgrades: Version changes should be intentional with full regression testing

Bug ID #L004 [Fixed]

Use Ownable2Step

Vulnerability Type
Missing Best Practices

Severity
Low

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://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/e6a0b0e049b673543a682f0019862aa00fa96395/contracts/CapxRewardDistributor.sol#L85

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 in b0ece48

Bug ID #G001 [Won't Fix]

Cheaper Inequalities in require()

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
The contract was found to be performing comparisons using inequalities inside the require statement. When inside the require statements, non-strict inequalities (>=, <=) are usually costlier than strict equalities (>, <).

Affected Code

- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L411-L411
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L500-L500
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L563-L563
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L572-L573
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L641-L641
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L647-L648
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L653-L654
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L746-L746
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L771-L771
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L909-L909
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1160-L1160

Impacts
Using non-strict inequalities inside "require" statements costs more gas.

Remediation
It is recommended to go through the code logic, and, **if possible**, modify the non-strict inequalities with the strict ones to save gas as long as the logic of the code is not affected.

Retest
Client's comment: We prioritize code clarity and maintainability over micro-optimizations that save <0.001% gas.

Bug ID #G002 [Won't Fix]

Cheaper conditional operators

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
Upon reviewing the code, it has been observed that the contract uses conditional statements involving comparisons with unsigned integer variables. Specifically, the contract employs the conditional operators x != 0 and x > 0 interchangeably. However, it's important to note that during compilation, x != 0 is generally more cost-effective than x > 0 for unsigned integers within conditional statements.

Affected Code

- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L481-L481
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L558-L558
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L750-L750
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L820-L822
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L861-L861
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1156-L1156
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1160-L1160
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1164-L1164
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1168-L1168

Impacts

Employing x != 0 in conditional statements can result in reduced gas consumption compared to using x > 0. This optimization contributes to cost-effectiveness in contract interactions.

Remediation
Whenever possible, use the x != 0 conditional operator instead of x > 0 for unsigned integer variables in conditional statements.

Retest
Client's comment: While != 0 is marginally cheaper than > 0 for unsigned integers (~3 gas), we've chosen not to implement this optimization.

Bug ID #G003 [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://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L483-L486
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L487-L490
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L492-L495
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L499-L502
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L557-L560
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L566-L569
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L571-L575
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L646-L650
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L745-L748
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L749-L752

- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1143-L1146
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1147-L1150
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1151-L1154
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1155-L1158
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1159-L1162
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1163-L1166
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1167-L1170

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 fixed in b0ece48

Bug ID #G004 [Won't Fix]

Splitting Require/Revert Statements

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
Require/Revert statements when combined using operators in a single statement usually lead to a larger deployment gas cost but with each runtime calls, the whole thing ends up being cheaper by some gas units.

Affected Code

- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1159-L1162
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/e6a0b0e049b673543a682f0019862aa00fa96395/contracts/CapxRewardDistributor.sol#L173
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/e6a0b0e049b673543a682f0019862aa00fa96395/contracts/CapxRewardDistributor.sol#L410-L411

Impacts
The multiple conditions in one **require/revert** statement combine require/revert statements in a single line, increasing deployment costs and hindering code readability.

Remediation
It is recommended to separate the **require/revert** statements with one statement/validation per line.

Retest
Client's comment: Keep combined conditions for runtime efficiency
Use short-circuit evaluation
Custom errors provide better savings than splitting

Bug ID#G005 [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.

Affected Code

- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L576
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L657
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1084
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1112

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 in b0ece48

Bug ID #G006 [Won't Fix]

Cheaper Inequalities in if()

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
The contract was found to be doing comparisons using inequalities inside the "if" statement. When inside the "if" statements, non-strict inequalities (>=, <=) are usually cheaper than the strict equalities (>, <).

Affected Code

- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1077
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/7fb707ed51bf6b2d4b2fa69d5dd0976da9ee1e95/contracts/PreSale.sol#L1105
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/e6a0b0e049b673543a682f0019862aa00fa96395/contracts/CapxRewardDistributor.sol#L316
- https://github.com/Capx-AI/Capx-Launchpad-Contracts/blob/e6a0b0e049b673543a682f0019862aa00fa96395/contracts/CapxRewardDistributor.sol#L321

Impacts
Using strict inequalities inside "if" statements costs more gas.

Remediation
It is recommended to go through the code logic, and, **if possible**, modify the strict inequalities with the non-strict ones to save gas as long as the logic of the code is not affected.

Retest:
Client's comment: Non-strict inequalities (>=, <=) are marginally cheaper than strict (>, <) in if() statements (~3 gas).

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

Audited by CRED SHIELDS

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com