Audited by CRED SHIELDS

CredShields

Smart Contract Audit

Nov 7th, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Taco Studios between Oct 30th, 2025, and Oct 31st, 2025. A retest was performed on Nov 6th, 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

Taco Studios

Table of Contents

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

Taco Studios engaged CredShields to perform a smart contract audit from Oct 30th, 2025, to Oct 31st, 2025. During this timeframe, nine (9) vulnerabilities were identified. **A retest was performed on Nov 6th, 2025, and all the bugs have been addressed.**

During the audit, one (1) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Taco Studios" 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	4	2	1	1	**9**

Table: Vulnerabilities Per Asset in Scope

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

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

Taco Studios 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 Oct 30th, 2025, to Oct 31st, 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://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.sol

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

Taco Studios 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, nine (9) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
Owner Can Cause Unpack Reverts For Pack Holders	High	Denial of Service (SCWE-087)
User Can Manipulate Block Timing To Predict And Maximize Rewards	Medium	Business Logic (SC03-LogicErrors)
Unpack Can Be DoSed If Chunk Configuration Is Removed	Medium	Denial of Service (SCWE-087)
Owner Can Prevent Pack Unpacking For Users	Medium	Business Logic (SC03-LogicErrors)
Owner May Reduce User Rewards After Unpack Request	Medium	Business Logic (SC03-LogicErrors)
Missing _gap In Upgradable Contract	Low	Business Logic (SC03-LogicErrors)
Missing events in important functions	Low	Missing Best Practices

Outdated Pragma	Informational	Outdated Compiler Version (SCWE-061)
Gas Optimization in Increments	Gas	Gas optimization (SCWE-082)

Table: Findings in Smart Contracts

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

Taco Studios is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Nov 6th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Owner Can Cause Unpack Reverts For Pack Holders	High	Fixed [Nov 6th, 2025]
User Can Manipulate Block Timing To Predict And Maximize Rewards	Medium	Fixed [Nov 6th, 2025]
Unpack Can Be DoSed If Chunk Configuration Is Removed	Medium	Fixed [Nov 6th, 2025]
Owner Can Prevent Pack Unpacking For Users	Medium	Fixed [Nov 6th, 2025]
Owner May Reduce User Rewards After Unpack Request	Medium	Fixed [Nov 6th, 2025]
Missing _gap In Upgradable Contract	Low	Fixed [Nov 6th, 2025]
Missing events in important functions	Low	Fixed [Nov 6th, 2025]
Outdated Pragma	Informational	Fixed [Nov 6th, 2025]
Gas Optimization in Increments	Gas	Fixed [Nov 6th, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #H001 [Fixed]

Owner Can Cause Unpack Reverts For Pack Holders

Vulnerability Type
Denial of Service (SCWE-087)

Severity
High

Description
The adminWithdraw() function is intended to allow the contract owner to withdraw stuck NFTs that are present in a pack's reward pool and transfer them to a recipient. Only the owner may call adminWithdraw(), and the function accepts a packCollection parameter used to locate and remove the reward from boosterConfigs[packCollection].rewardsPool. The contract also tracks availability using the global mapping isRewardAvailable[collection][tokenId].

The implementation first checks isRewardAvailable and then attempts to remove the reward by scanning only boosterConfigs[packCollection].rewardsPool. If the owner passes a packCollection that does not actually contain the reward (e.g., a different pack), the function will still proceed to transfer the NFT from the contract and set isRewardAvailable[collection][tokenId] = false, but the reward entry remains in its actual pack's rewardsPool. The root cause is the mismatch between a global availability flag and pack-local removal logic combined with no validation that the supplied packCollection actually contains the reward.

Affected Code

- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.sol#L381

Impacts
Users opening packs can encounter reverts during unpack because the rewards pool may still contain entries whose NFTs were already transferred and marked unavailable; selected indices pointing to those stale entries will trigger RewardAlreadyClaimed() or otherwise break unpack. This can deny pack holders the ability to open packs and claim rewards, causing service disruption and loss of user trust.

Remediation

Require adminWithdraw to verify the specified packCollection actually contains the reward and only perform the transfer after removing the reward from that pack's pool; if the reward is not found in the provided pack, revert to avoid transferring and desynchronizing state.

Retest

This issue has been fixed by removing the adminwithdraw functionality.

Bug ID #M001 [Fixed]

User Can Manipulate Block Timing To Predict And Maximize Rewards

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
In the unpack() flow, reward selection indices are derived from mixedSeed = uint256(keccak256(abi.encodePacked(randomSeed, blockhash(block.number - 1), msg.sender))). The value blockhash(block.number - 1) is publicly known when the user submits the transaction, making the entire mixedSeed fully predictable at submission time.

Because the randomness depends on a past block hash, users can compute the expected reward outcomes for each block and selectively broadcast unpack transactions only when the resulting indices correspond to rare or high-value NFTs. This allows users to cherry-pick favorable outcomes without any verifiable randomness or commit-reveal mechanism, breaking the intended fairness of the booster pack system. The root cause is the reliance on deterministic, user-controlled inputs (msg.sender, randomSeed, and a known block hash) for randomness generation.

Affected Code
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.sol#L302

Impacts
Users can deterministically time transactions to draw rare or valuable rewards, claiming the rewards unfairly and disadvantagering honest users.

Remediation
Replace the blockhash-derived seed with a verifiable, unpredictable randomness source (e.g., Chainlink VRF) or a commit-reveal / signer-reveal scheme so the final seed cannot be known by the caller before submitting unpack, and ensure the signature covers the final randomness.

Retest
This issue has been fixed.

Bug ID #M002 [Fixed]

Unpack Can Be DoSed If Chunk Configuration Is Removed

Vulnerability Type
Denial of Service (SCWE-087)

Severity
Medium

Description
The clearPackConfigChunked() function allows the contract owner to remove a portion of rewards from a pack configuration using cfg.rewardsPool.pop(). This operation correctly updates the reward pool size but does not update boosterConfigs[packCollection].nftPerPack. The nftPerPack value determines how many NFTs each pack yields during unpack.

If the owner removes rewards through clearPackConfigChunked without adjusting nftPerPack, the contract state becomes inconsistent: the remaining rewards pool can become smaller than nftPerPack. When users later attempt to open their packs, the unpack function will revert at runtime due to the if (poolSize < cfg.nftPerPack) revert NotEnoughRewardsConfigured(); check, permanently blocking the last users from claiming their rewards.

Affected Code
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.sol#L198

Impacts
Remaining pack holders will be unable to open their booster packs once the reward pool becomes smaller than the nftPerPack value. This effectively breaks the reward distribution flow.

Remediation
Reduce nftPerPack proportionally when rewards are removed via clearPackConfigChunked to maintain configuration consistency.

Retest
This issue has been fixed by setting collectionEnabled is false during maintenance, and reactivation is blocked unless distribution is exactly consistent via finalizePackConfig.

Bug ID #M003 [Fixed]

Owner Can Prevent Pack Unpacking For Users

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
The contract exposes clearPackConfigChunked for the owner to remove up to chunkSize entries from boosterConfigs[packCollection].rewardsPool; unpacking (userflow) requires collectionEnabled[packCollection] == true as enforced by if (!collectionEnabled[packCollection]) revert CollectionNotEnabled();. The intended behavior is that a collection remains enabled while rewards remain available and becomes disabled only when the rewards pool is fully cleared and the configuration removed.

clearPackConfigChunked unconditionally sets collectionEnabled[packCollection] = false; immediately after popping the requested chunk(s), regardless of whether cfg.rewardsPool.length became zero. The root cause is that the collection-enabled flag is cleared outside of the terminal condition that deletes the config and flips isInitialized[packCollection], breaking the invariant that enabled == rewards exist.

Affected Code

- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.sol#L228

Impacts
Alice buys a pack and the contract still holds remaining rewards, but because the owner called clearPackConfigChunked (removing a partial chunk) the collectionEnabled[packCollection] flag is false; Alice cannot call unpacking due to CollectionNotEnabled and her pack is effectively unopenable, causing stuck user funds/rewards and denial of expected service.

Remediation
Only set collectionEnabled[packCollection] = false when the rewards pool is actually empty; move the flag assignment into the branch that runs when cfg.rewardsPool.length == 0.

Retest

This issue is fixed by immediately setting collectionEnabled = false. This freezes user operations while the owner removes or adjusts rewards.

Bug ID #M004 [Fixed]

Owner May Reduce User Rewards After Unpack Request

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
When a user calls requestUnpack, their pack NFT is burned and a pending request is stored in pendingPackRequests. The rewards that will later be given are determined from boosterConfigs[packCollection] during the unpack call.

However, before the user completes unpack, the contract owner can call clearPackConfigChunked to remove or delete reward data from boosterConfigs. Because there is no lock or snapshot of the reward configuration at the time of requestUnpack, the owner's action can reduce or completely remove rewards available to that user.

Affected Code
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.sol#L245-270

Impacts
A user may burn their NFT and lose expected rewards if the owner clears or modifies the reward pool before unpack is called. This results in financial loss and breaks the fairness of the unpacking process.

Remediation
Lock or snapshot the reward configuration for a collection once a user requests an unpack to prevent further modification until the process is completed.

Retest
This issue has been fixed by adding a new mapping pendingPacks[packCollection] tracks the number of active unpack requests per booster collection.

Bug ID #L001 [Fixed]

Missing _gap In Upgradable Contract

Vulnerability Type
Business Logic (SC03-LogicErrors)

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://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.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 by adding a storage gap.

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://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.sol#L102-L105

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
Event is being emitted after important state change.

Bug ID #I001 [Fixed]

Outdated Pragma

Vulnerability Type
Outdated Compiler Version (SCWE-061)

Severity
Informational

Description
The smart contract is using an outdated version of the Solidity compiler specified by the pragma directive i.e. 0.8.28. 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

- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.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 0.8.29 pragma version.
Reference: https://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-061/

Retest
-

Bug ID#G001 [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://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/a3ee68f715bfd624b15cf1f17abf155e9c929cb7/launchpad/contracts/OkidoriLaunchpad.sol#L347

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 is 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

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