Audited by CRED SHIELDS

CredShields

Smart Contract Audit

August 6th, 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 July 31st, 2025, and August 1st, 2025. A retest was performed on August 5th, 2025.

Author

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

Reviewers

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

Prepared for

Taco Studios

Table of Contents

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

Taco Studios engaged CredShields to perform a smart contract audit from July 31st, 2025, to August 1st, 2025. During this timeframe, 12 vulnerabilities were identified. **A retest was performed on August 5th, 2025, and all the bugs have been addressed.**

During the audit, 3 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
Okidori	2	1	4	3	0	2	**12**
	2	**1**	**4**	**3**	**0**	**2**	**12**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Okidori'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 the Okidori 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 July 31st, 2025, to August 1st, 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/

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, 12 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
Collection Owners and Sellers Receive Incorrect Royalty Amounts Due to Flawed Fee Calculation Logic	Critical	Business Logic Issue (SCWE-001)
Malicious Collection Owners Can Set Excessive Royalty Rates That Effectively Steal Seller Proceeds	Critical	Missing Input Validation (SC04-Lack Of Input Validation)
Sellers Can Create Fraudulent ERC1155 Listings and Sell Tokens Externally While Maintaining Active Marketplace Listings	High	Business Logic Issue (SCWE-001)
ERC1155 Token Holders Cannot Create Multiple Listings for Different Quantities of the Same Token ID	Medium	Business Logic Issue (SCWE-001)
Denial Of Service To Legitimate Sellers Through Unbounded Array Iteration	Medium	Dependency on Block Gas Limit (SCWE-032)

Attacker can front-run initialization to gain unauthorized ownership of the marketplace contract	Medium	Insufficient Protection Against Front-Running (SCWE-037)
Marketplace owner can redirect royalty payments to themselves for any NFT collection	Medium	Business Logic Issue (SCWE-001)
Buyer can accidentally overpay by sending ETH during ERC20 token purchases	Low	Missing Input Validation (SC04-Lack Of Input Validation)
Missing events in important functions	Low	Missing Best Practices
Missing zero address validations	Low	Missing Input Validation (SC04-Lack Of Input Validation)
Cheaper conditional operators	Gas	Gas optimization (SCWE-082)
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 August 5th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Collection Owners and Sellers Receive Incorrect Royalty Amounts Due to Flawed Fee Calculation Logic	Critical	Fixed [Aug 4, 2025]
Malicious Collection Owners Can Set Excessive Royalty Rates That Effectively Steal Seller Proceeds	Critical	Fixed [Aug 4, 2025]
Sellers Can Create Fraudulent ERC1155 Listings and Sell Tokens Externally While Maintaining Active Marketplace Listings	High	Fixed [Aug 4, 2025]
ERC1155 Token Holders Cannot Create Multiple Listings for Different Quantities of the Same Token ID	Medium	Fixed [Aug 4, 2025]
Denial Of Service To Legitimate Sellers Through Unbounded Array Iteration	Medium	Fixed [Aug 4, 2025]
Attacker can front-run initialization to gain unauthorized ownership of the marketplace contract	Medium	Fixed [Aug 5, 2025]
Marketplace owner can redirect royalty payments to themselves for any NFT collection	Medium	Fixed [Aug 4, 2025]
Buyer can accidentally overpay by sending ETH during ERC20 token purchases	Low	Fixed [Aug 4, 2025]
Missing events in important functions	Low	Fixed [Aug 4, 2025]

Missing zero address validations	Low	Fixed [Aug 4, 2025]
Cheaper conditional operators	Gas	Fixed [Aug 4, 2025]
Gas Optimization in Increments	Gas	Fixed [Aug 4, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #C001 [Fixed]

Collection Owners and Sellers Receive Incorrect Royalty Amounts Due to Flawed Fee Calculation Logic

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The Okidori marketplace contract contains a critical mathematical error in the _splitPriceExact function that incorrectly calculates royalty distributions. The vulnerability stems from the improper calculation of royalty amounts based on the net seller amount rather than the total purchase price.

In the current implementation, the function first calculates the net amount that should go to the seller by dividing the final price by the sum of basis points plus total fees. Subsequently, it calculates the royalty amount by applying the royalty basis points to this already-reduced net amount. This approach is fundamentally flawed because royalties should be calculated as a percentage of the total transaction value, not as a percentage of the seller's portion after fees have been deducted.

The erroneous calculation occurs in lines where netAmount = (finalPrice * BPS_DENOM) / (BPS_DENOM + totalFeeBps) is computed first, followed by royaltyAmount = (netAmount * royaltyBps) / BPS_DENOM. This creates a compounding reduction effect where the royalty base is artificially diminished by the marketplace fee before the royalty percentage is applied. The mathematical relationship becomes distorted because the royalty is calculated on a subset of the intended base amount, effectively reducing the creator's compensation below the agreed-upon percentage of the sale price.

Affected Code

- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L498

Impacts

Collection owners and creators receive significantly less royalty compensation than intended, with the shortfall becoming more pronounced as marketplace fees increase.

Remediation

The corrected implementation should calculate the marketplace fee as `feeAmount = (finalPrice * feeBps) / BPS_DENOM` and the royalty amount as `royaltyAmount = (finalPrice * royaltyBps) / BPS_DENOM`. The net amount paid to the seller would then be determined by subtracting both the fee and royalty from the final price: `netAmount = finalPrice - feeAmount - royaltyAmount.`

Retest

This issue has been fixed by changing the logic `_splitPriceExact()` so that the owner and seller receive the correct amount.

Bug ID #C002 [Fixed]

Malicious Collection Owners Can Set Excessive Royalty Rates That Effectively Steal Seller Proceeds

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

Severity
Critical

Description
The Okidori marketplace contract contains a critical vulnerability in its royalty management system that allows collection owners to set arbitrarily high royalty percentages without any upper bounds validation. The vulnerability manifests in the setCollectionRoyalties function, which permits collection owners to specify royalty amounts in basis points without implementing reasonable maximum thresholds or validation checks.

The problematic logic resides in the function where collection ownership is verified through IOwnable(contractAddress).owner() == _msgSender(), followed by a direct call to _setCollectionRoyalties without any validation of the royalty amount parameter. The contract accepts any uint256 value for the royalty basis points, meaning a malicious collection owner could set royalties to 10,000 basis points (100%) or even higher values that exceed the total transaction amount.

This unrestricted royalty setting capability becomes particularly dangerous when combined with the fee calculation logic in _splitPriceExact. The function calculates the total fee basis points as totalFeeBps = feeBps + royaltyBps and then determines the net amount using netAmount = (finalPrice * BPS_DENOM) / (BPS_DENOM + totalFeeBps). When royalty basis points approach or exceed 10,000, the denominator becomes disproportionately large, resulting in negligible net amounts for sellers

Affected Code
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L153

Impacts

Malicious collection owners can exploit this vulnerability to appropriate seller proceeds by setting royalty rates at or near 100%, effectively converting legitimate sales transactions into mechanisms for wealth extraction.

Remediation

The primary fix involves implementing strict maximum royalty limits within the setCollectionRoyalties and _setCollectionRoyalties functions.

Retest

This issue has been fixed by setting maximum royalty limits in _setCollectionRoyalties function.

Bug ID #H001 [Acknowledged]

Sellers Can Create Fraudulent ERC1155 Listings and Sell Tokens Externally While Maintaining Active Marketplace Listings

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
High

Description
The Okidori marketplace contract contains a vulnerability in its ERC1155 token listing mechanism that allows sellers to create and maintain active listings for tokens they no longer own. The vulnerability arises from the contract's reliance on ownership and approval validation only at the time of listing creation, without implementing adequate safeguards to ensure continued token availability throughout the listing's lifecycle.

The problematic logic manifests in the listNFT function, where ERC1155 token ownership is validated using `nft1155.balanceOf(_msgSender(), tokenId) >= 1` and approval is confirmed through `nft1155.isApprovedForAll(_msgSender(), address(this))`. However, once these initial checks pass and the listing is created, the contract does not implement any mechanism to lock the tokens in escrow or continuously monitor the seller's token balance. This creates a window of opportunity where sellers can legitimately list their ERC1155 tokens and subsequently transfer or sell them through external channels while maintaining active marketplace listings.

Affected Code
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L203

Impacts
The most direct impact involves buyers when attempting to purchase ERC1155 tokens that sellers no longer possess. When a buyer initiates a purchase transaction for a stale listing, the transaction will fail during the token transfer phase.
From a systemic perspective, the prevalence of stale listings could significantly degrade user experience and platform reputation. Buyers who encounter frequent transaction failures due to invalid listings may lose confidence in the marketplace, leading to reduced trading volume and liquidity.

Remediation
The fix should be to lock the tokens while listing.

Retest
This issue has been acknowledged by the team.

Bug ID #M001 [Acknowledged]

ERC1155 Token Holders Cannot Create Multiple Listings for Different Quantities of the Same Token ID

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Medium

Description
The Okidori marketplace contract contains a fundamental design flaw in its listing validation logic that incorrectly treats ERC1155 tokens with identical semantics to ERC721 tokens, preventing users from creating multiple listings for different quantities of the same token ID. The vulnerability manifests in the listNFT function where the duplicate listing check performs a simplistic comparison that only considers contract address and token ID, completely ignoring the quantity-based nature of ERC1155 tokens.

The problematic logic resides in the duplicate prevention mechanism where the contract iterates through existing seller listings and rejects any new listing that matches both existingListing.contractAddress == contractAddress and existingListing.tokenId == tokenId. This approach fundamentally misunderstands the ERC1155 standard, which allows users to own multiple units of the same token ID and should permit them to list different quantities at various price points to maximize market efficiency.

The flawed validation occurs regardless of whether the seller owns sufficient tokens to support multiple listings. For instance, a user who owns 100 units of token ID #5 should be able to create separate listings for 10 units at 1 ETH each, 20 units at 0.9 ETH each, and 30 units at 0.8 ETH each to implement a quantity-based pricing strategy. However, the current implementation prevents this legitimate use case by blocking any subsequent listing attempts after the first listing is created.

Affected Code

- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L220

Impacts

This vulnerability significantly constrains the economic utility of the marketplace for ERC1155 token holders and creates artificial limitations that reduce market efficiency and user satisfaction. The impact extends beyond individual user inconvenience to encompass broader marketplace competitiveness and adoption challenges.
ERC1155 token holders are unable to implement sophisticated pricing strategies that would naturally emerge in efficient markets.

Remediation
The vulnerability requires a comprehensive redesign of the duplicate listing validation logic to properly accommodate the quantity-based nature of ERC1155 tokens while maintaining necessary safeguards against overselling and invalid listings. The remediation strategy must balance user flexibility with market integrity and technical robustness.

Retest
This issue has been acknowledged by the team.

Bug ID #M002 [Fixed]

Denial Of Service To Legitimate Sellers Through Unbounded Array Iteration

Vulnerability Type
Dependency on Block Gas Limit (SCWE-032)

Severity
Medium

Description
The Okidori marketplace contract contains a critical denial of service vulnerability in the listNFT function that stems from an unbounded loop iteration over the listingsOfSeller array. When a user attempts to list an NFT, the contract performs a linear search through all existing listings for that seller to prevent duplicate listings of the same NFT. This check is implemented in lines where the function iterates through sellerListings.length without any upper bound constraints.

As sellers accumulate listings over time, the listingsOfSeller array grows indefinitely. The contract fails to implement any mechanism to limit the array size or provide efficient lookup methods. Each iteration loads a Listing struct from storage, performs memory allocation, and executes comparison operations, resulting in linear gas cost growth proportional to the number of existing listings.

Affected Code
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L218

Impacts
As the number of listings per seller increases, the gas cost for the listNFT function grows linearly, eventually exceeding block gas limits and preventing legitimate sellers from creating new listings.

Remediation
Replace the current duplicate checking logic with a mapping that tracks active listings by a composite key of contract address and token ID.

Retest
This issue has been fixed by implementing a mapping that tracks active listings by a composite key of contract address and token ID.

Bug ID #M003 [Fixed]

Attacker can front-run initialization to gain unauthorized ownership of the marketplace contract

Vulnerability Type
Insufficient Protection Against Front-Running (SCWE-037)

Severity
Medium

Description
The Okidori marketplace contract contains a critical initialization vulnerability that allows malicious actors to gain unauthorized ownership through front-running attacks. The contract implements the UUPS (Universal Upgradeable Proxy Standard) pattern with an upgradeable architecture, but fails to properly secure the initialization process against front-running attacks.
The vulnerability stems from the absence of the _disableInitializers() call in the contract constructor and the public accessibility of the initialize() function without proper access controls.

When the proxy contract is deployed, there exists a time window between deployment and the intended initialization call where any external actor can invoke the initialize() function. The initializer modifier from OpenZeppelin only ensures the function can be called once, but does not restrict who can make this initial call. During deployment, if the legitimate deployer's initialization transaction is not mined first, an attacker can submit a competing transaction with higher gas fees to front-run the initialization process.

Affected Code
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L136

Impacts
An attacker who successfully front-runs the initialization gains full ownership privileges and the protocol owner can never again call initialize().

Remediation
The primary remediation involves adding the _disableInitializers() call within the contract constructor to prevent initialization of the implementation contract itself.

Retest

The bug has been fixed by adding `_disableInitializers()` in the constructor.

Bug ID #M004 [Acknowledged]

Marketplace owner can redirect royalty payments to themselves for any NFT collection

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Medium

Description
The Okidori marketplace contract contains a centralization vulnerability that grants the marketplace owner excessive control over royalty payments across all NFT collections traded on the platform. The vulnerability stems from the implementation of the setCollectionRoyaltiesOwner() function, which allows the marketplace owner to unilaterally override royalty configurations for any NFT collection without requiring consent from the original collection owner or creators.

Affected Code
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L169

Impacts
The exploitation of this vulnerability enables the marketplace owner to systematically redirect royalty payments away from legitimate NFT creators and collection owners, resulting in significant financial harm to the creative community.

Remediation
Eliminate the setCollectionRoyaltiesOwner() function entirely if no legitimate administrative use case exists for marketplace-controlled royalty overrides.

Retest
This issue has been acknowledged by the team.

Bug ID #L001 [Fixed]

Buyer can accidentally overpay by sending ETH during ERC20 token purchases

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

Severity
Low

Description
The Okidori marketplace contract contains an input validation vulnerability in the buyNFT() function that fails to verify that msg.value is zero when processing ERC20 token payments. The vulnerability arises from incomplete payment method validation logic that allows users to accidentally send both ETH and ERC20 tokens simultaneously during a single purchase transaction.

Affected Code

- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L337

Impacts
The vulnerability primarily results in accidental financial loss for buyers who mistakenly send ETH during ERC20 token purchases.

Remediation
Add a require(msg.value == 0, "No ETH required for ERC20 payment") check at the beginning of the ERC20 payment processing branch, immediately after the else statement.

Retest
This issue has been fixed by adding a check that msg.value == 0 when payment is in form of ERC20 tokens.

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/main/marketplace/contracts/Okidori.sol?ref_type=heads#L386

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 adding the event FeeBasisPointsSet.

Bug ID #L003 [Fixed]

Missing zero address validations

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

Severity
Low

Description:
The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Code

- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L180
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L203
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L419
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L429
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L462

Impacts
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest
This issue has been fixed by adding zero address validation.

Bug ID #G001 [Fixed]

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://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L209
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L332
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L344

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
This issue has been fixed by changing the conditional operator.

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.

Affected Code

- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L218
- https://gitlab.com/taco-wax/taiko-marketplace/sc-marketplace-audit/-/blob/main/marketplace/contracts/Okidori.sol?ref_type=heads#L469

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 by changing for loop based to mapping based.

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