Audited by CRED SHIELDS

CredShields

Limbo Smart Contract Audit

December 31, 2024 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Allin Gaming between October 29th, 2024, and December 17th, 2024. A retest was performed on December 18th, 2024.

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

Reviewers
Aditya Dixit (Research Team Lead), Shreyas Koli (Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor)

Prepared for
Allin Gaming

Table of Contents

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

Allin Gaming engaged CredShields to perform a smart contract audit from October 29th, 2024, to December 17th, 2024. During this timeframe, 9 vulnerabilities were identified. **A retest was performed on December 18th, 2024, and all the bugs have been addressed.**

High and Critical vulnerabilities represent the greatest immediate risk to "Allin Gaming" and should be prioritized for remediation. No such issues were found during the audit.

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
Limbo	0	0	2	2	5	0	**9**
	0	**0**	**2**	**2**	**5**	**0**	**9**

Table: Vulnerabilities Per Asset in Scope

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

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

Allin Gaming engaged CredShields to perform the Limbo 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 October 29th, 2024, to December 17th, 2024, 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/AllInGaming1/casino/tree/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo

2.1.2 Documentation

The Allin Gaming's team provided documentation for all the assets in scope and promptly answered all our questions.

2.1.3 Audit Goals

CredShields employs a combination of in-house tools and manual methodologies to conduct thorough security audits for Rust-based smart contracts. The audit process primarily involves manually reviewing the contract's source code, following best practices for Rust and WebAssembly (Wasm) development, and leveraging an internally developed, industry-aligned checklist. The team focuses on understanding key concepts, creating targeted test cases, and analyzing business logic to identify potential vulnerabilities.

2.2 Retesting Phase

Allin Gaming 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 presents the results of the security assessment. Findings are organized by severity and categorized by asset, with references to relevant classifications or standards. Each asset section includes a summary for clarity. The executive summary table provides an overview of the total number of identified security vulnerabilities for each asset, grouped by risk level.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 9 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	Vulnerability Type
Missing address validation in instantiate()	Medium	Missing Input Validation
Missing validation for max_multiplier	Medium	Missing Input Validation
Truncation of betting reward	Low	Data Typecasting
Unchecked minimum bet amount	Low	Missing Input Validation
Missing ownership transfer mechanism	Informational	Insecure Ownership Transfer
Unnecessary use of to_string()	Informational	Code Optimization
Length comparison to zero	Informational	Code Optimization
Unneeded late initialization of a variable	Informational	Code Optimization
Unnecessary let binding in return statement	Informational	Code Optimization

Table: Findings in Smart Contracts

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

Allin Gaming is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on December 18th, 2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Missing address validation in instantiate()	Medium	**Fixed** [Dec 18, 2024]
Missing validation for max_multiplier	Medium	**Fixed** [Dec 18, 2024]
Truncation of betting reward	Low	**Fixed** [Dec 18, 2024]
Unchecked minimum bet amount	Low	**Fixed** [Dec 18, 2024]
Missing ownership transfer mechanism	Informational	**Fixed** [Dec 18, 2024]
Unnecessary use of to_string()	Informational	**Fixed** [Dec 18, 2024]
Length comparison to zero	Informational	**Fixed** [Dec 18, 2024]
Unneeded late initialization of a variable	Informational	**Fixed** [Dec 18, 2024]
Unnecessary let binding in return statement	Informational	**Fixed** [Dec 18, 2024]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Missing address validation in instantiate()

Vulnerability Type
Missing Input Validation

Severity
Medium

Description
The instantiate function fails to validate the owner, bank_addr, and random_addr fields provided in the InstantiateMsg. This lack of validation not only permits improperly formatted or invalid addresses but also introduces issues with address representation, particularly related to case sensitivity. In the Cosmos SDK and CosmWasm ecosystem, addresses are expected to conform to the Bech32 format, which is case-insensitive. However, inconsistencies may arise when addresses are provided in uppercase during instantiation but are later used in lowercase by functions such as info.sender.

For example, if the owner address is stored in uppercase, subsequent checks comparing it to info.sender(which always provides addresses in lowercase) may fail. This can lead to the contract incorrectly identifying the owner, potentially bypassing administrative controls or locking out legitimate users.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L42
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L44
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L45

Impacts

The inconsistency between uppercase and lowercase address representations could result in failed access control checks, preventing legitimate users or administrators from performing actions, such as managing bets or updating contract settings.

Remediation
To address these issues, the instantiate() function should implement validation for all addresses provided in the InstantiateMsg.

Retest
This issue has been fixed as per the recommendations.

Bug ID #2 [Fixed]

Missing validation for max_multiplier

Vulnerability Type
Missing Input Validation

Severity
Medium

Description
The initialization of the game_config structure in the instantiate function does not validate the max_multiplier value provided in the InstantiateMsg. Specifically, there is no check to ensure that the max_multiplier is greater than zero. If a zero or otherwise invalid value is provided, it could lead to undefined or undesirable behavior in the game logic. For instance, a zero max_multiplier may result in payouts being incorrectly calculated or cause the game to malfunction entirely. This omission creates a risk of exploitable or unintended game configurations being accepted during instantiation.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L61

Impacts
An invalid max_multiplier could render the game unplayable or lead to incorrect payouts, resulting in financial losses or dissatisfaction among users.

Remediation
The instantiate function should validate the max_multiplier value to ensure it is greater than zero and within a sensible range

Retest
This issue has been fixed as per the recommendations.

Bug ID #3 [Fixed]

Truncation of betting reward

Vulnerability Type
Data Typecasting

Severity
Low

Description

In the betting logic, the user's winnings are calculated based on their bet amount and multiplier, producing a value, scaled_result, represented as Uint256. This value is then typecast to Uint128 using Uint128::try_from(scaled_result). If scaled_result exceeds the maximum value of Uint128, the conversion will truncate the higher-order bits. In such cases, the contract assigns the fallback value Uint128::MAX. However, the conditions under which scaled_result would exceed Uint128 are highly improbable, requiring extraordinarily large bet amounts or multipliers, well beyond typical operational scenarios.

While this truncation could theoretically lead to incorrect or oversized payouts, it is an edge case unlikely to occur during normal usage. The impact is further mitigated by practical constraints on bet sizes and multipliers in real-world deployments.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L412

Impacts
This scenario is rare and does not present a realistic concern for most users. As such, it has minimal impact on the integrity or usability of the betting system.

Remediation
While the issue poses no significant risk, adding an optional validation step to ensure scaled_result fits within the Uint128 range before conversion can enhance robustness.

Retest
This issue has been fixed by introducing validation for scaled_result fits within the Uint128 range.

Bug ID #4 [Fixed]

Unchecked minimum bet amount

Vulnerability Type
Missing Input Validation

Severity
Low

Description
The update_min_bet_amount() functions lack validation for the amount parameter. This allows for the possibility of setting MIN_AMOUNT to zero, which is illogical in the context of betting.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L200

Impacts
Allowing an invalid MIN_AMOUNT could result in a betting environment where users need clarification about the minimum required bet.

Remediation
It is recommended to implement zero validation for the amount parameter while updating the minimum bet amount.

Retest
This issue has been fixed as per the recommendations.

Bug ID #5 [Fixed]

Missing ownership transfer mechanism

Vulnerability Type
Insecure Ownership Transfer

Severity
Informational

Description
The contract sets the initial administrator (owner) via the instantiate function during deployment. However, it does not provide any functionality to transfer ownership or update the admin after deployment. This creates a design limitation as the ownership cannot be changed, even if there is a need to transfer it to a new owner.

Ownership transfer is an essential feature for decentralized systems to ensure flexibility and recoverability. If the current admin's private keys are lost or compromised, the inability to transfer ownership could lead to the permanent loss of control over the contract.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs

Impacts
If the admin loses access to their private keys or if the keys are compromised, there is no way to transfer ownership to a new secure address, leading to a permanent loss of control over the contract.

Remediation
It is recommended to implement a two-step ownership transfer in the contract.

Example code :

```
// Execute: Transfer Ownership
pub fn execute_transfer_ownership(
        deps: DepsMut,
        _env: Env,
        info: MessageInfo,
        new_admin: String,
) -> Result<Response, ContractError> {
```

```
        let admin = ADMIN.load(deps.storage)?;

        if info.sender != admin {
        return Err(ContractError::OnlyAdmin {});
        }

        if new_admin.is_empty() {
        return Err(ContractError::EmptyNewAdmin {});
        }

        PENDING_ADMIN.save(deps.storage, &Some(new_admin.clone()))?;
        Ok(Response::new().add_attribute("action",
"transfer_ownership").add_attribute("pending_admin", new_admin))
}

// Execute: Accept Ownership
pub fn execute_accept_ownership(
        deps: DepsMut,
        _env: Env,
        info: MessageInfo,
) -> Result<Response, ContractError> {
        let pending_admin = PENDING_ADMIN.load(deps.storage)?;

        if pending_admin.is_none() || pending_admin.as_ref().unwrap() !=  &info.sender.to_string()
{
        return Err(ContractError::OnlyPendingAdmin {});
        }

        ADMIN.save(deps.storage, &info.sender.to_string())?;
        PENDING_ADMIN.save(deps.storage, &None)?;

        Ok(Response::new().add_attribute("action",
"accept_ownership").add_attribute("new_admin", info.sender.to_string()))
}
```

Retest

This issue has been fixed by adding new functions: transfer_admin_control() and accept_admin_control().

Bug ID #6 [Fixed]

Unnecessary use of `to_string()`

Vulnerability Type
Code Optimization

Severity
Informational

Description

The code unnecessarily converts the `msg.owner` value to a `String` using `.to_string()` when its reference can be used directly with `.as_ref()`. This pattern introduces minor overhead by creating an unnecessary allocation for a new `String`, which could be avoided. `msg.owner.to_string()` can be replaced with `msg.owner.as_ref()`, as `msg.owner` already implements the required traits. While this issue does not affect functionality, it slightly impacts performance and readability.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L82

Impacts
The impact of this issue is minimal and primarily affects code clarity and efficiency.

Remediation
Replace `.to_string()` with `.as_ref()` in the affected line to eliminate the unnecessary allocation.

Retest
This issue has been fixed as per the recommendations.

Bug ID #7 [Fixed]

Length comparison to zero

Vulnerability Type
Code Optimization

Severity
Informational

Description
In the contract, the comparison of the length of a collection (info.funds) to zero can be replaced with the more idiomatic and expressive .is_empty() method. The .is_empty() method is not only clearer but also more concise and semantically explicit. It directly communicates the intent of checking whether the collection is non-empty, enhancing readability without introducing any functional changes.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L149
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L172

Impacts
This issue does not affect functionality or performance but impacts code clarity. Using .is_empty() makes the code easier to read and understand, particularly for developers familiar with idiomatic Rust.

Remediation
Replace .len() != 0 with !is_empty() wherever applicable. The updated line will be:

```
-   if !info.funds.len() != 0 {
+   if !info.funds.is_empty() {
```

Retest
This issue has been fixed as per the recommendations.

Bug ID #8 [Fixed]

Unneeded late initialization of a variable

Vulnerability Type
Code Optimization

Severity
Informational

Description
The identified issue pertains to unnecessary late initialization of the multiplier variable in the limbo smart contract module. The variable is declared without an initial value and subsequently assigned later in the code. This practice does not pose a direct security risk but reduces code clarity and maintainability.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/contract.rs#L370
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/query.rs#L104

Impacts
While the current implementation functions as intended, deferring initialization unnecessarily introduces potential for confusion or inadvertent errors during future code modifications. A more streamlined and clear approach would involve declaring and initializing the variable in a single step.

Remediation
To resolve this, the initialization of the multiplier should be combined with its declaration.

Retest
This issue has been fixed as per the recommendations.

Bug ID #9 [Fixed]

Unnecessary let binding in return statement

Vulnerability Type
Code Optimization

Severity
Informational

Description
The current code uses an unnecessary let binding to store the result of an operation before returning it. The variable truncated is assigned a value and then immediately returned. This pattern introduces an extra step that does not improve clarity or functionality. The operation can be returned directly, making the code more concise and readable.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/limbo/src/helpers.rs#L158-L159

Impacts
The impact of this issue is minimal and does not affect the functionality or performance of the code. However, the unnecessary use of let adds an extra line of code and makes the logic less direct, which could reduce readability, especially for simpler expressions.

Remediation
Return the result of the operation directly, eliminating the need for the let binding.

Retest
This issue has been fixed as per the recommendations.

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