Audited by CRED SHIELDS

CredShields

Classic Dice 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, 7 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. 1 Critical issue was 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
Classic Dice	1	0	1	2	3	0	**7**
	1	**0**	**1**	**2**	**3**	**0**	**7**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Classic Dice'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 Classic Dice 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/classic-dice

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, 7 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	Vulnerability Type
Incorrect validation in submit_bet() leads to stuck funds in the contract	Critical	Denial of Service
Incorrect range of bet.number	Medium	Business Logic Issue
Admin may lose funds while updating random_addr or bank_addr	Low	Missing Input Validation
Unchecked minimum bet amount	Low	Missing Input Validation
Missing Ownership Transfer Mechanism	Low	Insecure Ownership Transfer
Unnecessary let binding may lead to less efficient code	Informational	Code Optimization
Unnecessary return statement increases code verbosity	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
Incorrect validation in submit_bet() leads to stuck funds in the contract	Critical	**Fixed** [Dec 18, 2024]
Incorrect range of bet.number	Medium	**Not Fixed** [Dec 18, 2024]
Admin may lose funds while updating random_addr or bank_addr	Low	**Fixed** [Dec 18, 2024]
Unchecked minimum bet amount	Low	**Fixed** [Dec 18, 2024]
Missing Ownership Transfer Mechanism	Low	**Fixed** [Dec 18, 2024]
Unnecessary let binding may lead to less efficient code	Informational	**Fixed** [Dec 18, 2024]
Unnecessary return statement increases code verbosity	Informational	**Fixed** [Dec 18, 2024]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Incorrect validation in submit_bet() leads to stuck funds in the contract

Vulnerability Type
Denial of Service

Severity
Critical

Description
The submit_bet() function allows users to place bets with the number of rolls. However, the validation for the number of rolls is incorrectly implemented. The condition in the submit_bet() function does not handle cases where the user submits a bet with auto_bet.rolls equal to 0. As a result, the validation does not trigger an error, allowing the invalid roll count of 0 to be recorded in the contract. Later, when resolving bets using the resolve_bet() function, the calculation is performed while calculating bet_amount_per_roll in resolve_bet(), dividing initial_balance by auto_bet.rolls. Here dividing by 0 will cause panic in the contract, leading to all bets in the same batch failing to resolve. User funds associated with these bets will remain locked in the contract indefinitely, causing a Denial of Service (DoS) condition.

Affected Code
- https://github.com/AllInBetsCom/casino/blob/main/contracts/classic-dice/src/helpers.rs#L52-L58
- https://github.com/AllInBetsCom/casino/blob/main/contracts/classic-dice/src/contract.rs#L345

Impacts
The resolve_bet() function will panic when attempting to divide by zero, effectively halting the resolution of bets for all users. The user's funds will remain stuck in the contract indefinitely, causing financial loss.

Remediation

It is recommended to fix the validation in the validate_bet() function and update the validation logic to ensure that auto_bet.rolls cannot be 0.

Suggested fix for validate_bet() :

```
if auto_bet.rolls <= 1 || auto_bet.rolls > MAX_ROLLS {
        return Err(ContractError::InvalidRollCountForClass {
        class: "Advanced".to_string(),
        rolls: auto_bet.rolls,
        });
}
```

Retest

This issue has been fixed as per the recommendations.

Bug ID #2 [Won 't Fix]

Incorrect range of bet.number

Vulnerability Type
Business Logic Issue

Severity
Medium

Description
The betting contract classic-dice restricts the range of bet.number to 3–97, excluding the rare-probability numbers 2 and 98. This limitation prevents players from accessing higher multipliers (49x) associated with 2 and 98, forcing them to settle for lower multipliers (33x) at 3 and 97. Consequently, the bank misses opportunities to profit from frequent player losses on these high-risk bets, disrupting the risk-reward dynamics.

Affected Code
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/classic-dice/src/helpers.rs#L30-L42

Impacts
Players lose access to high-reward options, diminishing the fairness and appeal of the game. The bank faces reduced profitability as it cannot capitalize on rare-probability bets, creating financial inefficiencies.

Remediation
Set DELTA_VALUE to 1 to correctly validate the range as 2–98. This simple adjustment restores fairness and ensures the game functions as intended.

Retest
Client's comment: We will be keeping the range from 3-97. There's nothing wrong in this if we want to have higher house edge.

Bug ID #3 [Fixed]

Admin may lose funds while updating random_addr or bank_addr

Vulnerability Type
Missing Input Validation

Severity
Low

Description
The update_bank_address() and update_random_address() functions in the contract are restricted to admin access and allow updates to critical contract state. However, these functions do not validate the info.funds field, which represents any tokens sent alongside the transaction. If the admin inadvertently sends funds when invoking these functions, the tokens will remain locked in the contract's balance.

Affected Code
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/classic-dice/src/contract.rs#L99-L115
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/classic-dice/src/contract.rs#L120-L138

Impacts
If the admin mistakenly sends funds during the execution of these functions, the tokens will become irretrievable and locked in the contract.

Remediation
To prevent accidental fund loss, the contract should explicitly validate the info.funds field in update_bank_address() and update_random_address() functions like other functions.

Retest
This issue has been fixed as per the recommendations.

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. The absence of validation 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/classic-dice/src/contract.rs#L154

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

Remediation
It is recommended to implement 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/main/contracts/classic-dice/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 let binding may lead to less efficient code

Vulnerability Type
Code Optimization

Severity
Informational

Description
The classic-dice smart contract contains an unnecessary let binding when returning the result of an expression. In the following code, the result of the multiplication is assigned to a variable reward before being returned:

```
let unused_wager = wager * unused_rolls;
unused_wager
```

While this is valid, it introduces an unnecessary step. The value can be returned directly without the intermediate let binding. This is a trivial inefficiency, as the same result could be achieved without the extra line of code.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/classic-dice/src/contract.rs#L193-L194
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/classic-dice/src/helpers.rs#L161-L163

Impacts
Although this inefficiency does not introduce a functional bug or security vulnerability, it contributes to unnecessary verbosity in the codebase. In cases of more complex expressions, unnecessary intermediate bindings may make the code harder to read and maintain. While the impact is minimal in this case, it is generally a good practice to return expressions directly when possible.

Remediation
To address this, the value can be returned directly without assigning it to an intermediate variable. The optimized code would look like this:

```
wager * unused_rolls;
```

Retest
This issue has been fixed as per the recommendations.

Bug ID #7 [Fixed]

Unnecessary return statement increases code verbosity

Vulnerability Type
Code Optimization

Severity
Informational

Description
The classic-dice contract contains an unnecessary return statement when returning Ok(response). In Rust, the return keyword is not needed when returning the final expression from a function. The return keyword can be omitted, and the expression can be returned directly. This results in more concise and cleaner code.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/classic-dice/src/contract.rs#L520
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/classic-dice/src/helpers.rs#L199

Impacts
The redundant return statement does not cause any functional or security issues, but it contributes to unnecessary verbosity in the code. This excess complexity can make the code harder to read and maintain, even though the impact is minor.

Remediation
To streamline the code and remove unnecessary verbosity, the return statement can be omitted, and the expression can be returned directly. The optimized code is as follows:

```
Ok(response);
```

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