Audited by CRED SHIELDS

CredShields

Coin Flip 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. 2 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
Coin Flip	1	1	0	1	4	0	**7**
	1	**1**	**0**	**1**	**4**	**0**	**7**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Coin Flip'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 Coin Flip 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/coin_flip

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 storage key handling in resolve_bet() causes the record to overwrite and accounting issues	High	Data Integrity Issue
Timing variability in curve25519-dalek's Scalar29::sub/Scalar52::sub	Low	Outdated Cargo Crate Version
Inefficient string allocations may lead to degraded contract performance	Informational	Code Optimization
Unnecessary let binding may lead to less efficient code	Informational	Code Optimization
Unnecessary return statement increases code verbosity	Informational	Code Optimization
Redundant conversion increases code complexity	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 storage key handling in resolve_bet() causes the record to overwrite and accounting issues	High	**Fixed** [Dec 18, 2024]
Timing variability in curve25519-dalek's Scalar29::sub/Scalar52::sub	Low	**Not Fixed** [Dec 18, 2024]
Inefficient string allocations may lead to degraded contract performance	Informational	**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]
Redundant conversion increases code complexity	Informational	**Fixed** [Dec 18, 2024]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Incorrect validation in submit_flip_coin() leading to stuck funds in contract

Vulnerability Type
Denial of Service

Severity
Critical

Description
The submit_flip_coin() function allows users to place bets for both single and multiple flips. However, the validation for the number of flips is incorrectly implemented. The condition in the submit_flip_coin() function does not handle cases where the user submits a bet with auto_bet.flips equal to 0, Classification Advanced, and GameType Multiple. As a result, the validation does not trigger an error, allowing the invalid flip count of 0 to be recorded in the contract with GameType equal to Multiple. Later, when resolving bets using the resolve_bet() function, the calculation is performed while calculating bal_for_single_flip in resolve_bet(), dividing initial_balance by auto_bet.flips. 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/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/coin_flip/src/contract.rs#L168-L173
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/coin_flip/src/contract.rs#L442

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 submit_flip_coin() function and update the validation logic to ensure that auto_bet.flips cannot be 0.

Suggested fix for submit_flip_coin() :

```
if auto_bet.flips <= 1 || auto_bet.flips > MAX_FLIPS {
        return Err(ContractError::InvalidFlipCountForClass {
        class: "Advanced".to_string(),
        flips: auto_bet.flips,
        });
}
```

Retest

This issue has been fixed as per the recommendations.

Bug ID #2 [Fixed]

Incorrect storage key handling in resolve_bet() causes the record to overwrite and accounting issues

Vulnerability Type
Data Integrity Issue

Severity
High

Description
The submit_flip_coin() function records user bets by incrementing the bet_id as state.bet_id += 1; The bet_id is then stored in the bet_queue, which uniquely identifies each bet. In the resolve_bet() function, while resolving bets, single bets are resolved and saved in the GAME storage using the bet_id as the key. However, for multiple bets, the resolve_bet() function stores the results in GAME using the global state.bet_id as the key instead of using the unique bet.bet_id. This results in overwriting records for each iteration of resolved multiple bets, the state.bet_id remains the same for that batch of multiple bets, causing the previously saved record to be overwritten. also, resolution records of previous bets are lost as each iteration overwrites the same key in the GAME storage. This will create accounting issues where the GAME storage no longer contains accurate records of resolved bets, leading to data integrity issues and potential mismatches when querying results.

Affected Code
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/coin_flip/src/contract.rs#L566

Impacts
Records of multiple bets are overwritten in the GAME storage. Contract administrators and users cannot verify the resolution of previous multiple bets, leading to accountability and transparency issues.

Remediation
It is recommended to use the unique bet.bet_id for storing the results of both single and multiple bets in the GAME storage.

```
GAME.save(deps.storage, bet.bet_id, &game)?;
```

Retest

This issue has been fixed as per the recommendations.

Bug ID #3 [Won't Fix]

Timing variability in curve25519-dalek's Scalar29::sub/Scalar52::sub

Vulnerability Type
Outdated Cargo Crate Version

Severity
Low

Description
The curve25519-dalek crate version 3.2.0 contains a vulnerability in its Scalar29::sub and Scalar52::sub methods that exhibit timing variability. This variability can be exploited by attackers to perform timing attacks, enabling them to deduce information about the cryptographic operations being performed. Such timing attacks are particularly concerning in cryptographic contexts.

Affected Code
- Cargo.lock #L353 (Available after compiling the code)

Impacts
If an attacker is able to exploit this timing variability, they could potentially recover private keys or other sensitive information that relies on the security of these subtraction operations.

Remediation
It is recommended to upgrade the curve25519-dalek crate to version 4.1.3 or later.

Retest
Client's Comment: This crate is not being used by us anywhere for cryptography so nothing sensitive is at stake.

Bug ID #4 [Fixed]

Inefficient string allocations may lead to degraded contract performance

Vulnerability Type
Code Optimization

Severity
Informational

Description

The coin_flip smart contract includes unnecessary calls to the to_string() method when invoking addr_validate for msg.bank_addr and msg.random_addr. The to_string() function creates a new String object unnecessarily, as the addr_validate function can directly accept a string slice (&str).

For instance, the current implementation:

```
deps.api.addr_validate(&msg.bank_addr.to_string())?;
```

can be replaced with the more efficient:

```
deps.api.addr_validate(msg.bank_addr.as_ref())?;
```

This avoids redundant memory allocations while maintaining functionality. Such inefficiencies, when repeated across multiple transactions, can lead to increased resource consumption.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/coin_flip/src/contract.rs#L56
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/coin_flip/src/contract.rs#L57

Impacts
Frequent use of inefficient string allocations may degrade contract performance, particularly under heavy load. Increased memory usage and processing times could make the contract slower to execute and less responsive in high-demand scenarios, potentially affecting overall user experience.

Remediation
To address this issue, replace the unnecessary calls to to_string() with as_ref(). The optimized implementation is as follows:

```
bank_addr: deps.api.addr_validate(msg.bank_addr.as_ref())?,
random_addr: deps.api.addr_validate(msg.random_addr.as_ref())?,
```

Adopting this change prevents redundant allocations, aligns with Rust's best practices, and improves overall contract efficiency.

Retest
This issue has been fixed as per the recommendations.

Bug ID #5 [Fixed]

Unnecessary let binding may lead to less efficient code

Vulnerability Type
Code Optimization

Severity
Informational

Description
The coin_flip 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 reward = wager * multiplier_actual;
reward
```

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/coin_flip/src/contract.rs#L107-L108

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 * multiplier_actual
```

Retest

This issue has been fixed as per the recommendations.

Bug ID #6 [Fixed]

Unnecessary return statement increases code verbosity

Vulnerability Type
Code Optimization

Severity
Informational

Description
The coin_flip 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/coin_flip/src/contract.rs#L613

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.

Bug ID #7 [Fixed]

Redundant conversion increases code complexity

Vulnerability Type
Code Optimization

Severity
Informational

Description
The coin_flip smart contract includes an unnecessary type conversion in the following line of code:

```
if max_possible_payout > max_wager.into() {
```

Here, the .into() method is used to convert max_wager to the same type, cosmwasm_std::Uint128, which is already the type of max_possible_payout. This conversion is redundant and does not change the behaviour of the code.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/coin_flip/src/helpers.rs#L45

Impacts
This redundant conversion increases the complexity of the code without providing any benefit. It introduces an unnecessary operation that does not affect the outcome, making the code less elegant and potentially harder to maintain.

Remediation
To streamline the code, simply remove the .into() conversion. The simplified code would be:

```
if max_possible_payout > max_wager {
```

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