Audited by

CredShields

Lottery 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, 4 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 vulnerabilities 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**
Lottery	2	0	1	0	1	0	**4**
	2	**0**	**1**	**0**	**1**	**0**	**4**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Lottery'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 Lottery 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/lottery

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, 4 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	Vulnerability Type
Incorrect argument variable passed for pay_out process	Critical	Business Logic Issue
Incorrect Handling of Max Reward in resolve_bet() Function Leading to Stuck Funds	Critical	Loss of Funds
Incorrect version comparison in the migrate function	Medium	Business Logic Issue
Missing ownership transfer mechanism	Informational	Insecure Ownership Transfer

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 argument variable passed for pay_out process	Critical	**Fixed** [Dec 18, 2024]
Incorrect Handling of Max Reward in resolve_bet() Function Leading to Stuck Funds	Critical	**Not Fixed** [Dec 18, 2024]
Incorrect version comparison in the migrate function	Medium	**Fixed** [Dec 18, 2024]
Missing ownership transfer mechanism	Informational	**Fixed** [Dec 18, 2024]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Incorrect argument variable passed for pay_out process

Vulnerability Type
Business Logic Issue

Severity
Critical

Description
In the resolve_bet() function pay_out() call is responsible for transferring the payout amount to the user when they win a bet. However, there is a logical error in the code where the pay_in_amount parameter is being incorrectly set to the payout value instead of the wager (i.e., the bet amount). Here, the pay_in_amount parameter should represent the initial wager amount placed by the player. Passing the payout value instead can result in paying an extra wager fee to the bank.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/lottery/src/contract.rs#L403

Impacts
Users will have to pay more fees than required since the wager fee is calculated based on the payout

Remediation
It is recommended to pass the initial bet amount instead of the total winnings i.e. payout variable.

Retest
This issue has been fixed by passing the correct parameter. I.e. game_config.ticket_price * bet.total_tickets.

Bug ID #2 [Not Fixed]

Incorrect Handling of Max Reward in resolve_bet( ) Function Leading to Stuck Funds

Vulnerability Type
Loss of Funds

Severity
Critical

Description
The resolve_bet() function is responsible for resolving bets from the bet queue and distributing payouts. During the resolution process. The function calculates the max_wager (maximum reward). It compares the user's payout with the max_wager. If the payout exceeds the max_wager, the payout is set to max_wager. Rewards are then distributed to the user. For subsequent bets in the queue, the max_wager is reset to zero, causing the payout for the next bet to also be set to zero. When the contract attempts to transfer the payout using pay_out(), the transaction fails due to the logic in pay_out() that disallows a zero amount and raises the error: "Total Amount cannot be zero". This failure causes the entire bet resolution process to revert, resulting in bets resetting back to their original state and users' funds remaining locked in the contract indefinitely.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/lottery/src/contract.rs#L396-L398

Impacts
The user's amounts remain stuck in the contract because bets cannot be resolved successfully.

Remediation
It is recommended to refund the user's bet if the payout amount is greater than max_wager.

Retest
This issue has not been resolved.

Bug ID #3 [Fixed]

Incorrect version comparison in the migrate function

Vulnerability Type
Business logic Issue

Severity
Medium

Description
In the migrate function, the comparison of contract versions relies on a string-based lexicographical comparison. This approach is flawed because string-based comparisons for versioning can produce incorrect results when dealing with semantic versioning (e.g., 1.10 would be considered less than 1.9 because the comparison is based on lexicographical order). This could allow inappropriate migrations or prevent legitimate ones, leading to unexpected contract behavior.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/lottery/src/contract.rs#L609-L624

Impacts
Migrations from a newer contract version to an older one could lead to incompatible state transitions or undefined behavior.

Remediation
It is recommended to use a semantic versioning library like Semver for Rust to handle version comparisons. This library can properly parse and compare version strings based on semantic versioning rules.

Retest
This issue has been fixed as per the recommendations.

Bug ID #4 [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/lottery/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
Implementing a two-step ownership transfer in the contract is recommended.

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().

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

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