Audited by

CredShields

Bank 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, 18 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. 6 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**
Bank	4	0	5	5	2	0	**16**
	4	**0**	**5**	**5**	**2**	**0**	**16**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the Bank'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 Bank 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/bank

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, 18 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	Vulnerability Type
Unauthorized liquidity withdrawal	Critical	Access Control
Large iteration data can cause DoS in resolve_bet()	Critical	Denial of Service (DoS)
Malicious users will prevent genuine users from adding liquidity	Critical	Denial of Service (DoS)
Incorrect reserve accounting	Critical	Business Logic Issue
Incorrect version comparison in the migrate function	Medium	Business Logic Issue
Missing validation in lp_limit during instantiate()	Medium	Missing Input Validation
Missing Validation for max_wager	Medium	Missing Input Validation
Missing functionality to toggle is_withdraw_enabled variable	Medium	Missing Functionality

Missing functionality to toggle is_deposit_enabled variable	Medium	Missing Functionality
Dead code	Low	Dead Code
Lack of removal functionality for whitelisted game addresses	Low	Missing Functionality
Missing address removal functionality of liquidity providers	Low	Missing Functionality
Missing whitelisted asset removal functionality	Low	Business Logic Issue
Missing info.funds validation	Low	Missing Input Validation
Unnecessary reference to the right operand	Informational	Code Optimization
Unnecessary let binding may lead to less efficient code	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
Unauthorized liquidity withdrawal	Critical	**Fixed** [Dec 18, 2024]
Large iteration data can cause DoS in resolve_bet()	Critical	**Not Fixed** [Dec 18, 2024]
Malicious users will prevent genuine users from adding liquidity	Critical	**Fixed** [Dec 18, 2024]
Incorrect reserve accounting	Critical	**Fixed** [Dec 18, 2024]
Incorrect version comparison in the migrate function	Medium	**Fixed** [Dec 18, 2024]
Missing validation in lp_limit during instantiate()	Medium	**Not Fixed** [Dec 18, 2024]
Missing Validation for max_wager	Medium	**Fixed** [Dec 18, 2024]
Missing functionality to toggle is_withdraw_enabled variable	Medium	**Fixed** [Dec 18, 2024]
Missing functionality to toggle is_deposit_enabled variable	Medium	**Fixed** [Dec 18, 2024]
Dead code	Low	**Fixed** [Dec 18, 2024]
Lack of removal functionality for whitelisted game addresses	Low	**Partially Fixed** [Dec 18, 2024]

Missing address removal functionality of liquidity providers	Low	**Fixed** [Dec 18, 2024]
Missing whitelisted asset removal functionality	Low	**Partially Fixed** [Dec 18, 2024]
Missing info.funds validation	Low	**Fixed** [Dec 18, 2024]
Unnecessary reference to the right operand	Informational	**Fixed** [Dec 18, 2024]
Unnecessary let binding may lead to less efficient code	Informational	**Fixed** [Dec 18, 2024]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Unauthorized liquidity withdrawal

Vulnerability Type
Access Control

Severity
Critical

Description
The remove_liquidity() function in the pool_manager contract fails to validate that the caller (info.sender) is the same as the sender parameter provided in the function call. This lack of validation creates an access control vulnerability, allowing any unauthorized user to create withdrawal requests on their behalf.
The issue arises because the function relies solely on the sender parameter, which can be arbitrarily set by the caller. Since there is no check to ensure that the function is being called by the legitimate owner of the liquidity, a malicious actor can exploit this flaw to manipulate the withdrawal queue and initiate unauthorized requests for any user's deposits.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/pool_manager.rs#L122-L182

Impact
Unauthorized users can queue withdrawals for other users without their consent, disrupting the affected users' liquidity/PnL management.

Remediation
To address this issue, the contract should enforce strict access control by validating that the caller of the function is the same as the sender parameter. This can be implemented by comparing info.sender with sender at the beginning of the function.

Retest
This issue has been fixed by adding a validation of sender against info.sender.

Bug ID #2 [Not Fixed]

Large iteration data can cause DoS in resolve_bet( )

Vulnerability Type
Denial of Service (DoS)

Severity
Critical

Description
The provided resolve_bet() function processes multiple nested loops to handle operations related to surplus and deficit tokens, calculate user profit and loss (PnL), and resolve deposit and withdrawal queues. These nested iterations depend on the size of the whitelisted_addresses, surplus and deficit tokens, user deposits, and the withdrawal and deposit queues.
As these data structures grow due to increased user activity or system scale, the gas consumption required to process the function increases significantly. If the amount of data being processed is large enough, it can exceed the gas limit of the transaction, causing the function to fail and revert. This issue occurs organically as a result of the contract's scalability limitations, without the need for malicious actors.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/pool_manager.rs#L194-L509

Impact
Liquidity providers are directly affected as their withdrawal/deposit requests remain unresolved when the function fails. Additionally, user profit and loss (PnL) calculations will also be impacted due to unprocessed updates during epochs where the function fails to execute due to gas exhaustion.

Remediation
It is recommended to restructure the resolve_bet() function to avoid processing large data structures in a single transaction.
Implementing batching mechanisms for processing surplus and deficit tokens, user deposits, and withdrawal queues can significantly reduce gas consumption. By dividing the resolution process into smaller, independent operations (different transactions), the contract can handle large datasets more efficiently without exceeding gas limits.

Retest

The current implementation calls the separated functions internally. Hence the gas used will be the same as before.

It is recommended to call all the separated functions individually and make sure:

1. Only authorized users can call those functions.
2. All the functions are invoked properly and in a timely manner, since the PnL, and pool size are dependent on them.

Bug ID #3 [Fixed]

Malicious users will prevent genuine users from adding liquidity

Vulnerability Type
Denial of Service (DoS)

Severity
Critical

Description
The add_liquidity() function in the pool manager contract enforces a limit on the number of liquidity providers (lp_limit). If the number of liquidity providers (lp_count) exceeds this limit, further attempts to add liquidity are reverted. However, the function does not validate the amount of liquidity being added during the initial check for new liquidity providers. This oversight allows a malicious user to exploit the system by creating multiple small deposits using different addresses until the lp_limit is reached.
Once the limit is reached, genuine users attempting to add significant liquidity to the pool will be blocked, as the system will erroneously treat the pool as full. Furthermore, this can disrupt the economic balance of the pool, as the attacker's deposits may be negligible compared to the genuine liquidity expected from legitimate users.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/pool_manager.rs#L96

Impacts
This creates a **denial-of-service (DoS)** condition for genuine users who are unable to add liquidity despite having significant deposits, leading to frustration and loss of trust in the system.

Remediation
To mitigate this issue, the contract should enforce a minimum liquidity deposit amount for new liquidity providers. This ensures that only users adding meaningful contributions to the pool are registered as liquidity providers.

Retest
This issue has been fixed by adding a MIN_LP_AMOUNT validation for every liquidity provider.

Bug ID #4 [Fixed]

Incorrect reserve accounting

Vulnerability Type
Business Logic Issue

Severity
Critical

Description
The pay_in() and pay_out() functions in the vault contract have a logical flaw that causes wager fees to be added to the reserves twice. Specifically, in the pay_in() function, the collect_wager_fee() function already increments the wager_fee_reserves with the fee charged during its execution. However, after the profit is added to the available_funds, the pay_in() function adds the amount_after_fee to wager_fee_reserves again. This duplication occurs despite the fact that the user pays the wager fee only once.

Similarly, in the pay_out() function, the same issue is present. Regardless of whether the outcome results in a profit or loss, the wager_fee_reserves are incremented twice: once when the fee is collected and again when processing profits or losses. This leads to an overstatement of the reserves and introduces inconsistencies in accounting, as the reserves do not reflect the actual amount collected from users.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/vault.rs#L133
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/vault.rs#L305
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/vault.rs#L322

Impacts
This vulnerability will cause the reserves to show incorrect values, overstating the wager fee reserves.

Remediation
It is recommended to remove the additional wager_fee_reserves increment in both the pay_in() and pay_out() functions, leaving the fee collection solely to the collect_wager_fee() function.

Retest

This issue has been fixed by removing extra calculation of wager_fee_reserves.

Bug ID #5 [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/bank/src/contract.rs#L173-L176

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 by using Semver library.

Bug ID #6 [Not Fixed]

Missing validation in lp_limit during instantiate()

Vulnerability Type
Missing Input Validation

Severity
Medium

Description
In the instantiate() function, the lp_limit parameter, which determines the maximum number of liquidity providers allowed in the vault, is directly saved into storage without validation. This unchecked input introduces the risk of initializing the vault with an excessively low or high limit, which could adversely affect the platform's functionality.
Furthermore, there is no mechanism within the contract to modify the lp_limit after initialization.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/contract.rs#L64

Impacts
If the limit is set too low, legitimate users may be unable to join the vault as liquidity providers, restricting the pool's growth.

Remediation
To address this vulnerability, validation logic should be implemented in the instantiate function to enforce reasonable boundaries for the lp_limit. For example, it should ensure that the value is neither excessively low nor impractically high.

Retest
Client's Comment: lp_limit will be removed, so this won't be present in the final code.

Bug ID #7 [Fixed]

Missing Validation for max_wager

Vulnerability Type
Missing Input Validation

Severity
Medium

Description
The given functions are responsible for updating the max_wager_fee, primary_token_max_wager_fee, and wager_fee, which is a Decimal variable used to calculate transaction fees in the contract. However, the function lacks validation to ensure that the input value is less than one. The issue arises because the functions lack validation to ensure that the input values are less than or equal to 1. A Decimal value greater than 1 results in charging more than 100% in fees, which is illogical and could harm the user experience or create exploitable scenarios.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L101-L114
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L117-L130
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L136-L149

Impacts
Users may lose more amount due to invalid fee configurations. The logic of the contract becomes unreliable, potentially leading to accounting errors and disputes.

Remediation
It is recommended to introduce proper validation to ensure that all percentage-based variables are less than or equal to 1.

Retest
This issue has been fixed by adding a validation as recommended.

Bug ID #8 [Fixed]

Missing functionality to toggle is_withdraw_enabled variable

Vulnerability Type
Missing Functionality

Severity
Medium

Description
The contract lacks the functionality to enable or disable withdrawals dynamically. The is_withdraw_enabled flag is set during the instantiate, and cannot be modified after deployment, making the withdrawal status fixed.

Affected Code
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/pool_manager.rs#L71-L74

Impacts
The lack of a dynamic toggle for the is_withdraw_enabled flag prevents the contract owner from managing withdrawal permissions. This means if withdrawals need to be paused (e.g., during an emergency or security issue), the owner cannot disable them. Similarly, if withdrawals are disabled, the owner cannot re-enable them.

Remediation
It is recommended to implement a function for the owner to toggle the is_withdraw_enabled flag.

Retest
This issue has been fixed as recommended.

Bug ID #9 [Fixed]

Missing functionality to toggle is_deposit_enabled variable

Severity
Missing Functionality

Vulnerability Type
Medium

Description
The contract lacks the functionality to enable or disable deposits dynamically. The is_deposit_enabled flag is set during the instantiate, and cannot be modified after deployment, making the deposit status fixed.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/pool_manager.rs#L71-L74

Impacts
The lack of a dynamic toggle for the is_deposit_enabled flag prevents the contract owner from managing deposit permissions. This means if deposits need to be paused (e.g., during an emergency or security issue), the owner cannot disable them. Similarly, if deposits are disabled, the owner cannot re-enable them.

Remediation
It is recommended to implement a function for the owner to toggle the is_deposit_enabled flag.

Retest
This issue has been fixed as recommended.

Bug ID #10 [Fixed]

Dead code

Vulnerability Type
Dead Code

Severity
Low

Description
The whitelist_address_lp() function contains a misleading comment stating that whitelisting has been removed, while the function still adds addresses to LIQUIDITY_PROVIDERS.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L55-L66

Impacts
This does not impact the security aspect of the Smart contract but prevents confusion when the code is sent to other developers or auditors to understand and implement.
This reduces the overall size of the contracts and also helps in saving gas.

Remediation
The comment should be updated to match the current functionality or the whitelisting logic should be completely removed from the function. This will eliminate the dead code and prevent confusion, enhancing the clarity and maintainability of the contract.

Retest
This issue has been resolved by removing the whitelist_address_lp() function.

Bug ID #11 [Partially Fixed]

Lack of removal functionality for whitelisted game addresses

Vulnerability Type
Missing Functionality

Severity
Low

Description
The whitelist_address_games() function allows adding game addresses to a whitelist along with associated game names but lacks a mechanism to remove them. This absence restricts administrators from effectively managing the whitelist, leaving outdated or incorrect entries in the system indefinitely.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L70-L95

Impacts
The inability to remove whitelisted game addresses or names can lead to a buildup of obsolete or erroneous entries, creating unnecessary administrative overhead and operational inefficiencies.

Remediation
It is recommended to implement a remove_whitelist_address_games() function with robust authentication and validation to allow authorized removal of assets.

Retest
This issue is partially fixed by adding a function to remove whitelisted games.

However, it does not check if the address_game is being used by any games that haven't been resolved yet. If the address_game is removed from the whitelisting before resolving the bet, It can be a Denial Of Service.

Client's comment: This will need updates in game contracts later.

Bug ID #12 [Fixed]

Missing address removal functionality of liquidity providers

Vulnerability Type
Missing Functionality

Severity
Low

Description
The whitelist_address_lp() function allows adding liquidity provider addresses to a whitelist but lacks a mechanism to remove them. This limitation restricts administrators from managing the whitelist effectively, leaving obsolete or incorrect addresses permanently in the system.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs

Impacts
The inability to remove whitelisted addresses can lead to an accumulation of outdated or erroneous entries, increasing administrative burden and operational inefficiencies.

Remediation
It is recommended to implement a remove_whitelist_address_lp() function with robust authentication and validation to allow authorized removal of assets.

Retest
This issue has not been resolved.

Bug ID #13 [Partially Fixed]

Missing whitelisted asset removal functionality

Vulnerability Type
Business Logic Issue

Severity
Low

Description
The whitelist_asset() function allows adding assets to a whitelist but lacks a mechanism to remove them. This omission limits the administrator's ability to manage the whitelist effectively, permanently leaving outdated or incorrect entries in the system.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs

Impacts
The inability to remove whitelisted assets can lead to an accumulation of obsolete or incorrect entries, increasing administrative burden and the risk of operational errors.

Remediation
It is recommended to implement a remove_whitelist_asset() function with robust authentication and validation to allow authorized removal of assets.

Retest
This issue is partially fixed by adding a function to remove whitelisted assets.

However, a new issue arises as the function does not verify if there are remaining balance reserves or pool funds associated with the asset, potentially leading to inconsistencies or account-related issues.
Also it does not check if the asset is being used by any games that haven't been resolved yet. If the asset is removed from the whitelisting before resolving the bet, It can be a Denial Of Service.

Client's comment: This will need updates in game contracts later.

Bug ID #14 [Fixed]

Missing info.funds validation

Vulnerability Type
Missing Input Validation

Severity
Low

Description
Few 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/bank/src/utils.rs#L56-L66
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L70-L95
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L101-L114
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L117-L130
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L136-L149
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L153-L167

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 given functions like other functions.

Retest

This issue has been fixed by adding info.funds validation across the functions.

Bug ID #15 [Fixed]

Unnecessary reference to the right operand

Vulnerability Type
Code Optimization

Severity
Informational

Description
the equality check unnecessarily takes a reference of the right operand. While this does not directly introduce functional errors or security vulnerabilities, it can lead to inefficient code execution. The reference is superfluous since the variable is already a reference or can be compared directly without taking an additional reference.

Affected Code

- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/query.rs#L141
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/utils.rs#L84
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/vault.rs#L49
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/vault.rs#L197
-

Impacts
Although this inefficiency does not introduce a functional bug or security vulnerability, it contributes to unnecessary verbosity in the codebase.

Remediation
To resolve this issue, review all instances of equality checks in the codebase and ensure that no unnecessary references are used for operands.
Specifically, remove the & operator in comparisons like caller == &owner, changing it to caller == owner. This will align with Rust's best practices.

```
Some(owner) => Ok(caller == owner),
```

Retest

This issue has not been resolved.

Bug ID #16 [Fixed]

Unnecessary let binding may lead to less efficient code

Vulnerability Type
Code Optimization

Severity
Informational

Description
The smart contract contains an unnecessary `let` binding when returning the result of an expression. In the following code, the result of the expression is assigned to a variable before being returned. 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/bank/src/query.rs#L232-L233
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/external.rs#L63-L89
- https://github.com/AllInBetsCom/casino/blob/34b3423bc31dc3645b17e98e9a407665db0d0807/contracts/bank/src/external.rs#L41-L61

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.

Retest
This issue has not been resolved.

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