Audited by CRED SHIELDS

CredShields

# Smart Contract Audit

May 7th, 2025 • CONFIDENTIAL

### Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Power Couple Coin between April 16th, 2025, and April 22nd, 2025. A retest was performed on May 5th, 2025.


### Author

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

### Reviewers

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


### Prepared for

Power Couple Coin

# Table of Contents

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

Power Couple Coin engaged CredShields to perform a smart contract audit from April 16th, 2025, to April 22nd, 2025. During this timeframe, 18 vulnerabilities were identified. **A retest was performed on May 5th, 2025, and all the bugs have been addressed.**

During the audit, 7 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Power Couple Coin" and should be prioritized for remediation.

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 | Σ |
|---|---|---|---|---|---|---|
| Lottery Contracts | 6 | 1 | 3 | 5 | 3 | **18** |
| | **6** | **1** | **3** | **5** | **3** | **18** |

*Table: Vulnerabilities Per Asset in Scope*

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

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

Power Couple Coin engaged CredShields to perform a 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 April 16th, 2025, to April 22nd, 2025, 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 |
| --- |
| **Lottery Contracts - Compressed Zip File** |

### 2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

### 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

Power Couple Coin 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.

## 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 contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

## 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 |
|---|---|---|
| Malicious Player Can Bypass Entry Fee By Faking Vault | Critical | Missing Account Validation |
| Admin/Creator Can Increase Rewards Without Transferring Tokens | Critical | Missing Account Validation |
| Single User Can Exploit Full Reward Pool | Critical | Logical Issue |
| Authority Failure Can Block Prize Transfers to the Winner | Critical | Incorrect Signature |
| Malicious User Can Manipulate the Winner Selection | Critical | Missing Account Validation |
| Creator Can Divert Token Rewards to Unauthorized Account | Critical | Logical Error |
| Malicious User Can Block Legitimate Lottery Initialization | High | Seed Collision |
| Buyer Can Trigger Useless State Write and Fail Transaction | Medium | Gas Griefing |

| | | |
|---|---|---|
| Setting Expired Lottery End Time Can Lead To Funds Stuck | Medium | Missing Input Validation |
| Admin Can Illegitimately Participate in Their Own Lottery | Medium | Logical Issue |
| Lottery Account Can Be Closed to Reclaim Admin Rent Deposit | Low | Missing Best Practices |
| Unauthorized User Can Become Program Admin on First Initialization | Low | Missing Access Control |
| Use of transfer instead of transfer_checked | Low | Missing Best Practices |
| Missing toolchain Version In Anchor.toml | Low | Environment Misconfiguration |
| Missing Default Address Check Allows Unintended Creator Assignment | Low | Missing Input Validation |
| Unused Code Increases Maintenance Burden and Binary Size | Informational | Dead Code |
| Duplicate Program Configuration Functions Cause Redundancy | Informational | Code Duplication |
| Manual Default Implementation Can Be Simplified with Derive | Informational | Code Quality |

*Table: Findings in Smart Contracts*

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

Power Couple Coin is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities. **A retest was performed on May 5th, 2025, and all the issues have been addressed.**

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

| VULNERABILITY TITLE | SEVERITY | REMEDIATION STATUS |
|---|---|---|
| Malicious Player Can Bypass Entry Fee By Faking Vault | Critical | **Fixed** [May 5, 2025] |
| Admin/Creator Can Increase Rewards Without Transferring Tokens | Critical | **Fixed** [May 5, 2025] |
| Single User Can Exploit Full Reward Pool | Critical | **Fixed** [May 5, 2025] |
| Authority Failure Can Block Prize Transfers to the Winner | Critical | **Fixed** [May 5, 2025] |
| Malicious User Can Manipulate the Winner Selection | Critical | **Fixed** [May 5, 2025] |
| Creator Can Divert Token Rewards to Unauthorized Account | Critical | **Fixed** [May 5, 2025] |
| Malicious User Can Block Legitimate Lottery Initialization | High | **Fixed** [May 5, 2025] |
| Buyer Can Trigger Useless State Write and Fail Transaction | Medium | **Fixed** [May 5, 2025] |
| Setting Expired Lottery End Time Can Lead To Funds Stuck | Medium | **Fixed** [May 5, 2025] |
| Admin Can Illegitimately Participate in Their Own Lottery | Medium | **Fixed** [May 5, 2025] |
| Lottery Account Can Be Closed to Reclaim Admin Rent Deposit | Low | **Fixed** [May 5, 2025] |

| | | |
|---|---|---|
| Unauthorized User Can Become Program Admin on First Initialization | Low | **Fixed** [May 5, 2025] |
| Use of transfer instead of transfer_checked | Low | **Fixed** [May 5, 2025] |
| Missing toolchain Version In Anchor.toml | Low | **Fixed** [May 5, 2025] |
| Missing Default Address Check Allows Unintended Creator Assignment | Low | **Fixed** [May 5, 2025] |
| Unused Code Increases Maintenance Burden and Binary Size | Informational | **Fixed** [May 5, 2025] |
| Duplicate Program Configuration Functions Cause Redundancy | Informational | **Fixed** [May 5, 2025] |
| Manual Default Implementation Can Be Simplified with Derive | Informational | **Fixed** [May 5, 2025] |

*Table: Summary of findings and status of remediation*

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

Bug ID #1 [Fixed]

### Malicious Player Can Bypass Entry Fee By Faking Vault

**Vulnerability Type**
Missing Account Validation

**Severity**
Critical

**Description**
The lottery_token_vault is only checked for mutability, not for being a valid PDA derived from the lottery ID. This lets a player pass their own token account as the vault. When token::transfer is invoked, the entry fee is not actually transferred to a secure, protocol-owned vault.

**Affected Code**
- Lib.rs#L617

**Impacts**
A malicious actor can continuously join lotteries without paying the entry fee. This breaks the fairness of the lottery, undermines protocol revenue, and can be exploited at scale to extract value or manipulate outcomes.

**Remediation**
It is recommended to enforce PDA derivation on lottery_token_vault using seeds and bump, which will ensure only the correct vault owned by the protocol can receive funds.

```
+ #[account(mut, seeds = [VAULT_PREFIX, lottery_id.as_bytes()], bump)]
- #[account(mut)]
  pub lottery_token_vault: AccountInfo<'info>,
```

**Retest**
This issue has been fixed.

Bug ID #2 [Fixed]

## Admin/Creator Can Increase Rewards Without Transferring Tokens

**Vulnerability Type**
Missing Account Validation

**Severity**
Critical

**Description**
In fund_rewards, the lottery_token_vault is not validated to be a PDA owned by the lottery program. An attacker with admin or creator privileges can pass their own token account instead, triggering a fake transfer and inflating lottery.total_token_rewards without actually sending any tokens.

**Affected Code**
- Lib.rs#L590

**Impacts**
Admin can fake reward funding, deceiving users and auditors about available prizes while risking insolvency during actual payout.

**Remediation**
Validate lottery_token_vault is derived from a known PDA with the correct seeds and bump.

```
#[account(
    mut,
    seeds = [LOTTERY_PREFIX, lottery_id.as_bytes()],
    bump,
    constraint = lottery.winner.is_some() @ LotteryError::NoWinnerSelected,
    constraint = lottery.winner.unwrap() == player.key() @ LotteryError::NotWinner,
    constraint = matches!(lottery.status, LotteryStatus::WinnerSelected) @
LotteryError::InvalidLotteryState
+   close = lottery.creator
  )]
  pub lottery: Account<'info, LotteryState>,
```

**Retest**
This issue has been fixed.

Bug ID #3 [Fixed]

## Single User Can Exploit Full Reward Pool

**Vulnerability Type**
Logical Issue

**Severity**
Critical

**Description**
In buy_ticket, no restrictions prevent a single user from purchasing all tickets. Since prize_amount = total_token_rewards + total entry fees, the user effectively gets back their full spend plus the total_token_rewards. This incentivizes sybil-like behavior to guarantee winning, especially in small or underfunded lotteries.

**Affected Code**
- Lib.rs#L131

**Impacts**
An attacker can drain the entire reward pool by buying all tickets, causing loss of incentive for real participants and economic loss to the sponsor.

**Remediation**
Limit tickets per user or subtract an admin commission from total_token_pool before calculating prize_amount.

**Retest**
This issue has been fixed by adding MAX_TICKETS_PER_USER

Bug ID #4 [Fixed]

## Authority Failure Can Block Prize Transfers to the Winner

**Vulnerability Type**
Incorrect Signature

**Severity**
Critical

**Description**
The token::transfer call uses ctx.accounts.lottery as the signer for lottery_token_vault, but SPL tokens require the authority field to match the actual authority of the token account. Since lottery_token_vault is not controlled by lottery, the PDA signature derived from lottery will fail validation.

**Affected Code**
- Lib.rs#L356

**Impacts**
The winner cannot claim the prize despite fulfilling all conditions, leading to a critical failure in the core reward mechanism.

**Remediation**
Update vault authority to be the PDA derived from LOTTERY_PREFIX and lottery_id, or use the actual vault authority in the signer seeds.

**Retest**
This issue has been fixed.

Bug ID #5 [Fixed]

## Malicious User Can Manipulate the Winner Selection

**Vulnerability Type**
Missing Account Validation

**Severity**
Critical

**Description**
In the select_winner function, the randomness_account_data is passed as an unchecked AccountInfo without ownership, signature, or content validation. This allows any user to call select_winner with a forged randomness account that returns predictable values favorable to the caller. The contract currently lacks enforcement to ensure the randomness source was securely generated and tied to the lottery instance during initialization.

**Affected Code**
- Lib.rs#L642

**Impacts**
An attacker can submit forged randomness accounts to make sure their own address is selected as the winner, breaking fairness and enabling unauthorized token capture from the lottery vault.

**Remediation**
Bind the randomness account to the lottery instance during initialization and validate its address in SelectWinner.

```
+ #[account(
+    constraint = randomness_account_data.key() == lottery.randomness_account @
LotteryError::InvalidRandomnessAccount
+ )]
  pub randomness_account_data: AccountInfo<'info>,
```

**Retest**
This issue has been fixed by adding randomness_account check

Bug ID #6 [Fixed]

## Creator Can Divert Token Rewards to Unauthorized Account

**Vulnerability Type**
Logical Error

**Severity**
Critical

**Description**
In the Initialize instruction, lottery_token_vault is passed as an unchecked AccountInfo without verification or constraints. This allows a creator to specify any token account as the vault, including one under their control. Since the lottery will later deposit tokens into this account, this misconfiguration enables a creator to redirect token rewards to arbitrary addresses. The vault should be a program-derived address (PDA) controlled by the lottery account to ensure safe custody of assets.

**Affected Code**
- Lib.rs#L560

**Impacts**
Creators can initialize a lottery using their own token account as the vault, siphoning off all token rewards and compromising the integrity of the reward distribution.

**Remediation**
Derive lottery_token_vault using a PDA tied to the lottery account and assert it in the instruction to enforce program authority.

```
- pub lottery_token_vault: AccountInfo<'info>,
+ #[account(
+    mut,
+    seeds = [LOTTERY_PREFIX, lottery.key().as_ref()],
+    bump,
+    token::authority = lottery,
+    token::mint = mint
+ )]
+ pub lottery_token_vault: Account<'info, TokenAccount>,
```

**Retest**

This issue has been fixed.

Bug ID #7 [Fixed]

## Malicious User Can Block Legitimate Lottery Initialization

**Vulnerability Type**
Seed Collision

**Severity**
High

**Description**
The Initialize instruction derives the lottery account using a user-supplied lottery_id as a seed (seeds = [LOTTERY_PREFIX, lottery_id.as_bytes()]). This allows any actor to preemptively create the account with a chosen lottery_id, effectively blocking the legitimate admin or creator from initializing a lottery under that ID. Since the PDA is deterministic, once occupied, the admin cannot overwrite or reclaim it without coordination from the attacker.

**Affected Code**
- Lib.rs#L546

**Impacts**
A malicious user can permanently deny access to a desired lottery_id, preventing the admin from creating lotteries with predictable or branded identifiers, disrupting operations or user trust.

**Remediation**
Restrict lottery_id usage or add an additional signer-verified seed such as admin.key().as_ref() to ensure only authorized entities can initialize accounts for a given ID.

```
- seeds = [LOTTERY_PREFIX, lottery_id.as_bytes()],
+ seeds = [LOTTERY_PREFIX, admin.key().as_ref(), lottery_id.as_bytes()],
```

**Retest**
This issue has been fixed by adding admin key as well in seeds

Bug ID #8 [Fixed]

## Buyer Can Trigger Useless State Write and Fail Transaction

**Vulnerability Type**
Gas Griefing

**Severity**
Medium

**Description**
In buy_ticket, the lottery status is checked via get_status, which mutably updates the status if the lottery has ended. However, this update occurs *before* the status is validated by a require! condition. When get_status changes the status from Active to EndedWaitingForWinner, the subsequent check fails, and the transaction reverts. Since all state changes are rolled back on failure, the status update is also lost—causing the same logic to repeat on every call and waste compute units.

**Affected Code**
- Lib.rs#L497

**Impacts**
Any user can repeatedly call buy_ticket on an expired lottery, triggering state updates that are rolled back on failure, leading to unnecessary compute consumption and potential denial-of-service risks under compute budgets.

**Remediation**
Make get_status a read-only function that does not mutate state, and require the status be updated in a separate admin-controlled transaction.

```
- self.update_status(LotteryStatus::EndedWaitingForWinner);
+ return if current_time > self.end_time && matches!(current_status, LotteryStatus::Active){
+    LotteryStatus::EndedWaitingForWinner
+ } else {
+    current_status
+ };
```

**Retest**
This issue has been fixed.

Bug ID #9 [Fixed]

## Setting Expired Lottery End Time Can Lead To Funds Stuck

**Vulnerability Type**
Missing Input Validation

**Severity**
Medium

**Description**
The `end_time` parameter is directly stored without checking if it lies in the future. If set to a past timestamp, `get_status()` will consider the lottery expired right after initialization. Since `buy_ticket` checks for `LotteryStatus::Active`, user entries will be rejected.

**Affected Code**
- Lib.rs#L42

**Impacts**
Although users are not directly affected, any reward funds deposited into the lottery become inaccessible, resulting in locked protocol assets. The issue is likely due to misconfiguration rather than malicious intent.

**Remediation**
It is recommended to validate `end_time` against the current block time, which will prevent accidental locking of funds due to a misconfigured or expired lottery.

```
+ require!(end_time > Clock::get()?.unix_timestamp, LotteryError::EndTimeInPast);
lottery.end_time = end_time;
```

**Retest**
This issue has been fixed by comparing endTime > currentTime.

Bug ID #10 [Fixed]

## Admin Can Illegitimately Participate in Their Own Lottery

**Vulnerability Type**
Logical Issue

**Severity**
Medium

**Description**
The buy_ticket function includes a check that prevents the lottery creator from purchasing tickets, enforcing a fair-play restriction. However, it does not apply the same restriction to the admin, who has similar privileged control over the lottery lifecycle. Since lottery.admin can still buy tickets, this creates a loophole that undermines the fairness policy and opens the door to potential abuse by privileged actors.

**Affected Code**
- Lib.rs#L131

**Impacts**
The admin can enter the lottery despite having control privileges.

**Remediation**
Add a restriction that prevents the admin from participating in the lottery, consistent with the policy for the creator.

```
+ require!(
+    ctx.accounts.player.key() != ctx.accounts.lottery.admin,
+    LotteryError::AdminCannotParticipate
+ );
```

**Retest**
This issue has been fixed.

Bug ID #11 [Fixed]

## Lottery Account Can Be Closed to Reclaim Admin Rent Deposit

**Vulnerability Type**
Missing Best Practices

**Severity**
Low

**Description**
In the ClaimPrize context, the lottery account is marked as mut, and while no current instruction allows its closure, the account is susceptible to being closed in the future if not explicitly restricted. Since the rent deposit for the lottery account is typically paid by the lottery creator or admin, best practice dictates that only this party should be permitted to reclaim it.

**Affected Code**
- Lib.rs#L659

**Impacts**
The lottery creator will lose the rent amount.

**Remediation**
Close the account after the winner has claimed the prize and send the rent amount to the payer of the account.

**Retest**
This issue has been fixed.

Bug ID #12 [Fixed]

## Unauthorized User Can Become Program Admin on First Initialization

**Vulnerability Type**
Missing Access Control

**Severity**
Low

**Description**
The initialize_program_config function initializes the program_config account without verifying the identity of the caller. If the account does not yet exist, any actor can invoke this function and set themselves as the config.admin. The absence of a precondition to restrict who may perform this one-time initialization exposes the program to hostile takeover if deployed without pre-seeding this account.

**Affected Code**
- Lib.rs#L763

**Impacts**
An attacker can initialize the program config and assign themselves as admin.

**Remediation**
Restrict the instruction to be callable only if the transaction is signed by a known deploy-time authority.

**Retest**
This issue has been fixed.

Bug ID #13 [Fixed]

## Use Of transfer Instead of transfer_checked

**Vulnerability Type**
Missing Best Practices

**Severity**
Low

**Description**
The fund_rewards, buy_ticket, and claim_price functions use the transfer instruction to move tokens from a user's payment account to the vault payment account. However, transfer does not validate the token mint or decimal configuration, allowing transfers involving unintended token types or incorrect amounts due to misinterpreted decimals. Without enforcing these checks, users or integrators may inadvertently or maliciously trigger transfers under misconfigured conditions.

**Affected Code**
- Lib.rs#L83
- Lib.rs#L188
- Lib.rs#L392

**Impacts**
An incorrect mint or mismatched decimal setup could lead to users sending the wrong token or an inaccurate amount, resulting in underpayment, overpayment, or accounting inconsistencies.

**Remediation**
Replace all transfer calls with transfer_checked, which validates both the mint and the number of decimals.

```
- transfer(...)
+ transfer_checked(..., mint.key(), expected_decimals)
```

**Retest**
This issue has been fixed.

Bug ID #14 [Fixed]

## Missing toolchain Version In Anchor.toml

**Vulnerability Type**
Environment Misconfiguration

**Severity**
Low

**Description**
The Anchor.toml file lacks explicit anchor_version and solana_version declarations under the [toolchain] section. This omission can lead to version drift across different development environments, potentially introducing unexpected behavior, compilation errors, or inconsistencies during deployment. Without a fixed toolchain version, CI/CD pipelines and team members may inadvertently use incompatible versions of Anchor or Solana.

**Affected Code**
- Anchor.toml

**Impacts**
Builds and deployments may fail or behave inconsistently across machines, reducing reliability and increasing debugging overhead.

**Remediation**
Explicitly define the Anchor and Solana versions in the [toolchain] section of Anchor.toml.

```
[toolchain]
+ anchor_version = "0.31.0"
+ solana_version = "1.18.25"
```

**Retest**
This issue has been fixed.

Bug ID #15 [Fixed]

## Missing Default Address Check Allows Unintended Creator Assignment

**Vulnerability Type**
Missing Input Validation

**Severity**
Low

**Description**
In the initialize function, the creator_key is directly assigned to lottery.creator without validating that it is a valid, non-default public key. Solana's Pubkey::default() (all zeros) is a valid Pubkey value, but semantically meaningless and typically indicates uninitialized data. Without explicit validation, the contract may unintentionally store an invalid or placeholder value as the lottery creator.

**Affected Code**
- Lib.rs#L40

**Impacts**
lottery may be created with an invalid creator address, undermining traceability and potentially breaking logic that relies on a valid creator identity.

**Remediation**
Add a check to ensure creator_key is not the default Pubkey::default().

```
+ require!(creator_key != Pubkey::default(), LotteryError::InvalidCreatorAddress);
```

**Retest**
This issue has been fixed.

Bug ID #16 [Fixed]

## Unused Code Increases Maintenance Burden and Binary Size

**Vulnerability Type**
Dead Code

**Severity**
Informational

**Description**
The codebase includes functions, parameters, or modules that are never invoked either internally or externally. In Rust-based Solana programs, retaining such unused code can cause confusion during auditing or collaborative development. Although this does not affect security directly, it increases cognitive overhead, bloats the compiled binary, and may obscure critical logic paths. A clean codebase helps ensure clarity, maintainability, and efficiency, especially in production deployments.

**Affected Code**
- Lib.rs#L2
- Lib.rs#L6
- Lib.rs#L186
- Lib.rs#L362-L372
- Lib.rs#L377-L388
- Lib.rs#L420
- Lib.rs#L432

**Impacts**
Redundant code increases binary size and audit complexity, potentially leading to misunderstandings or oversights during review and higher deployment costs.

**Remediation**
Remove all unused functions, parameters, and modules from the codebase to ensure clarity and optimal build efficiency.

**Retest**
This issue has been fixed.

Bug ID #17 [Fixed]

## Duplicate Program Configuration Functions Cause Redundancy

**Vulnerability Type**
Code Duplication

**Severity**
Informational

**Description**
The functions initialize_program_config and update_allowed_token appear multiple times in the codebase. This redundancy can lead to confusion about the active implementation, increase audit complexity, and introduce subtle logic inconsistencies if one copy is modified independently. In smart contract development, especially with Rust and Anchor, maintaining a single source of truth for each function is critical for correctness and maintainability.

**Affected Code**
- Lib.rs#L763
- Lib.rs#L775

**Impacts**
Developers and auditors may misinterpret the correct logic, and future updates may affect only one version of the function, leading to unexpected behavior.

**Remediation**
Remove all duplicate implementations of initialize_program_config and update_allowed_token, retaining only the most complete and correct version.

**Retest**
This issue has been fixed.

Bug ID #18 [Fixed]

## Manual Default Implementation Can Be Simplified with Derive

**Vulnerability Type**
Code Quality

**Severity**
Informational

**Description**
The LotteryStatus enum implements the Default trait manually to return LotteryStatus::Active. However, this implementation is redundant as it can be automatically derived using #[derive(Default)] along with #[default] on the desired variant. Using derivation improves readability and aligns with Rust idioms, reducing boilerplate and the chance of inconsistent defaults during refactoring.

**Affected Code**
- Lib.rs#L131

**Impacts**
No functional risk, but the manual Default implementation adds unnecessary complexity and may be overlooked in future updates.

**Remediation**
Replace the manual implementation with a derived Default trait and annotate the default variant accordingly.

```
- impl Default for LotteryStatus {
-     fn default() -> Self {
-         LotteryStatus::Active
-     }
- }
+ #[derive(Default)]
+ pub enum LotteryStatus {
+     #[default]
+     Active = 0,
```

**Retest**

This issue has been fixed.

# 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