Ponzimon Protocol Security Audit Report

July 18, 2025

Contents

1 Introduction

1.1 About Ponzimon Protocol

1.2 Source Code

1.3 Revision History

2 Overall Assessment

3 Vulnerability Summary

3.1 Overview

3.2 Security Level Reference

3.3 Vulnerability Details

4 Appendix

4.1 About AstraSec

4.2 Disclaimer

4.3 Contact

1 Introduction

1.1 About Ponzimon Protocol

Ponzimon is a gamified staking and collecting platform on the Solana blockchain. Players acquire digital cards with unique attributes, purchase farms, and stake cards to earn token rewards from a global emissions pool. They can upgrade farms, open booster packs for random new cards, or recycle unwanted cards for potential upgrades. The platform uses a two-step commit-reveal scheme based on slot hashes for randomness and includes token burns, protocol fees, and referral incentives to sustain its economy.

The audit was performed via the Hyacinth platform.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/begreatfulforreal/ponzimon-program
- Commit: a74e9975e6a77044c4fa766b4604e6671c2867c1

This is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/begreatfulforreal/ponzimon-program
- Commit: a9bb431d5df5d92a220b8cd6b20c2b8bf52552e1

1.3 Revision History

Version	Date	Description
v1.0	July 18, 2025	Initial Audit

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Ponzimon protocol. Throughout this audit, we identified a total of 4 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	–	–	–	–
High	–	–	–	–
Medium	2	1	–	1
Low	1	–	–	1
Informational	–	–	–	–
Undetermined	1	1	–	–
Total	4	2	–	2

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~ Improper Reward Logic in recycle_cards_settle()

M-2 Potential Risks Associated with Centralization

~~L-1~~ Enhanced Sanity Checks in discard_card()

U-1 Possible Manipulated Random Number in recycle_cards_settle()

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and re-mediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [M-1] Improper Reward Logic in recycle_cards_settle()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
instructions.rs	Business Logic	Medium	Medium	Addressed

The recycle_cards_settle() function finalizes a player's card recycling attempt, using on-chain data to determine if their cards are successfully upgraded or lost. In its current sequence, the function calls update_pool() to refresh the global reward state and then immediately synchronizes the player's last_acc_tokens_per_hashpower to this latest state. This synchronization happens before the player's pending rewards for the elapsed period are calculated and paid out, resulting in the player's accumulated rewards for that period being foregone. Moreover, the settle_open_booster() function shares the same issue.

ponzimon-program-main - instructions.rs

```rust
pub fn recycle_cards_settle(ctx: Context<RecycleCardsSettle>) -> Result<()> {
    ...

    // Settle rewards before changing player state
    update_pool(gs, clock.slot);
    player.last_acc_tokens_per_hashpower = gs.acc_tokens_per_hashpower;

    // Extract recycled card data from pending action
    let (card_indices_array, card_count) = if let PendingRandomAction::Recycle {
        card_indices,
        card_count,
    } = player.pending_action
    {
        (card_indices, card_count)
    } else {
        return Err(PonzimonError::NoRecyclePending.into());
    };

    ...
}
```

Remediation Replace the update_pool() function with the settle_and_mint_rewards() function, which handles the full reward distribution before updating the player's state.

3.3.2 [M-2] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Medium	Acknowledged

The Ponzimon protocol grants significant protocol-wide control to a single privileged owner account, which is authorized to perform administrative actions. These include resetting individual user states, enabling or disabling core system functionality, manually updating pool states, and arbitrarily modifying system parameters. This concentration of power contradicts the protocol's decentralized ethos, exposing it to substantial risks. A compromised or malicious owner could unilaterally alter protocol behavior, misappropriate funds, or disrupt the sale process, jeopardizing user trust and system integrity.

ponzimon-program-main - lib.rs

```
pub fn reset_player(ctx: Context<ResetPlayer>) -> Result<()> {
    instructions::reset_player(ctx)
}
pub fn toggle_production(ctx: Context<ToggleProduction>, enable: bool) -> Result<()> {
    instructions::toggle_production(ctx, enable)
}
pub fn update_pool_manual(ctx: Context<UpdatePool>) -> Result<()> {
    instructions::update_pool_manual(ctx)
}
```

Remediation To mitigate centralization risks, consider implementing a multi-signature wallet or a decentralized governance mechanism to manage critical actions. Additionally, introduce a time-lock mechanism for sensitive actions to provide users with advance notice and the opportunity to react to changes. If full decentralization is not feasible, ensure the owner's role is transparently documented, and consider transferring ownership to a secure, community-controlled entity over time to align with decentralization principles.

Response By Team This issue has been acknowledged by the team.

3.3.3 [L-1] Enhanced Sanity Checks in discard_card()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
instructions.rs	Business Logic	Low	Low	Addressed

The discard_card() function allows a player to remove a card from their inventory. A specific interaction sequence leads to an unintended outcome. A player can first call recycle_cards_commit(), which records the indices of the cards to be processed. Before settling this action, the player can call discard_card() to discard a different card. This discard operation modifies the player's inventory array (line 616), causing the indices of subsequent cards to shift. When the recycle_cards_settle() transaction is executed, it operates on the stored indices, which now point to different cards than those initially selected, resulting in the wrong cards being recycled.

ponzimon-program-main - instructions.rs

```
612 pub fn discard_card(ctx: Context<DiscardCard>, card_index: u8) -> Result<()> {
613     ...
614
615     // Remove the card using the helper function
616     player.batch_remove_cards(&[card_index])?;
617
618     player.last_acc_tokens_per_hashpower = gs.acc_tokens_per_hashpower;
619
620     emit!(CardDiscarded {
621         player: player.key(),
622         card_index,
623     });
624
625     Ok(())
626 }
```

Remediation To ensure the integrity of pending operations, the discard_card() function could be enhanced by first verifying that the player has no pending operations before proceeding, thereby preventing state conflicts.

3.3.4 [U-1] Possible Manipulated Random Number in recycle_cards_settle()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
instructions.rs	Business Logic	High	Low	Acknowledged

The recycle_cards_settle() function generates a random outcome for card recycling using the hash of a future predetermined slot (reveal_slot). The reveal_slot is derived by adding a fixed delay to the player's commit_slot. While this introduces a time delay, the source of randomness — the hash of a single, predictable slot — is deterministic. An advanced participant or validator could potentially influence or predict the outcome by observing the commit_slot and knowing the hash of the future reveal_slot.

ponzimon-program-main - instructions.rs

```rust
pub fn recycle_cards_settle(ctx: Context<RecycleCardsSettle>) -> Result<()> {
    ...
    let reveal_slot = player.commit_slot + MIN_RANDOMNESS_DELAY_SLOTS;
    ...

    let data = sysvar_slot_history.try_borrow_data()?;
    let num_slot_hashes = u64::from_le_bytes(data[0..8].try_into().unwrap());
    let mut pos = 8;
    let mut found_hash = None;
    for _ in 0..num_slot_hashes {
        let slot = u64::from_le_bytes(data[pos..pos + 8].try_into().unwrap());
        pos += 8;
        let hash = &data[pos..pos + 32];
        if slot == reveal_slot {
            found_hash = Some(hash);
            break;
        }
        pos += 32;
    }

    let random_value = found_hash.ok_or(PonzimonError::SlotNotFound)?; // Or your preferred error
    ...
}
```

Remediation To address this vulnerability, the system could be adjusted to incorporate multiple, less predictable on-chain sources, such as combining the reveal_slot hash with other dynamic values like the current timestamp or leader-produced data, making the outcome significantly more difficult to anticipate.

Response By Team The team acknowledges this issue but, given the extremely high cost of manipulating the slot hash, which likely far exceeds the potential benefits, has decided to accept the risk.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://x.com/AstraSecAI