Preduck Games (Solana)

Security Audit Report

December 8, 2025

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

BugBlow utilized a widely adopted approach to performing a security audit. Below is a breakout of how our team was able to identify and exploit vulnerabilities.

1.2.1 Code and architecture review:

- Review the source code.
- Read the project's documentation.
- Review the project’s architecture.

Stage goals

- Build a good understanding of how the application works.

1.2.2 Check code against vulnerability checklist:

- Understand the project’s critical assets, resources, and security requirements.

- Manual check for vulnerabilities based on the auditor's checklist.
- Run static analyzers.

Stage goals

- Identify and eliminate typical vulnerabilities (gas limit, replay, flash-loans attack, etc.)

1.2.3 Threat & Attack Simulation:

- Analyze the project's critical assets, resources and security requirements.
- Exploit the found weaknesses in a safe local environment.
- Document the performed work.

Stage goals

- Identify vulnerabilities that are not listed in static analyzers that would likely be exploited by hackers.
- Develop Proof of Concept (PoC).

1.2.4 Report found vulnerabilities:

- Discuss the found issues with developers
- Show remediations.
- Write an initial audit report.

Stage goals

- Verify that all found issues are relevant.

1.2.5 Fix bugs & re-audit code:

- Help developers fix bugs.
- Re-audit the updated code again.

Stage goals

- Double-check that the found vulnerabilities or bugs were fixed and were fixed correctly.

1.2.4. Deliver final report:

- The Client deploys the re-audited version of the code
- The final report is issued for the Client.

Stage goals

- Provide the Client with the final report that serves as security proof.

Severity classification

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Vulnerabilities leading to assets theft, fund access locking, or any other loss of funds.
High	Vulnerabilities that can trigger contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Vulnerabilities that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Vulnerabilities that do not have a significant immediate impact and could be easily fixed.

1.3 Project Overview

Preduck Games project implements a decentralized prediction market on the Solana blockchain using the Anchor framework. It allows users to place bullish (price increase) or bearish (price decrease) bets on an asset's price movement (e.g., SOL/USD) during specific rounds. Each round progresses through distinct phases: betting, locked, and either completed or cancelled. Winners receive rewards proportional to their bet size, while a treasury account collects fees (e.g., 3% of

total bets). Administrators can configure market parameters, pause operations, and manage treasury funds.

1.4 Project Dashboard

Project Summary

Title	Preduck Games Security Audit
Client name	Preduck Games
Project name	Preduck Games Smart Contracts (Solana) https://preduck.games
Timeline	20.11.2025 - 01.12.2025
Number of Auditors	3

Project Last Log

Date	Commit Hash	Note
20.11.2025	635a1aeb153980af07dd97ec67601ead65437069	chore: added log

Project Scope

The audit covered the following smart contract files.

File name
lib.rs

events.rs
rrrors.rs
constants.rs
instructions/bettings.rs
instructions/claiming.rs
instructions/initialize.rs
instructions/managing.rs
instructions/mod.rs
instructions/round_management.rs
state/bet.rs
state/market.rs
state/mod.rs
state/nonce.rs
state/round.rs

1.5 Summary of findings

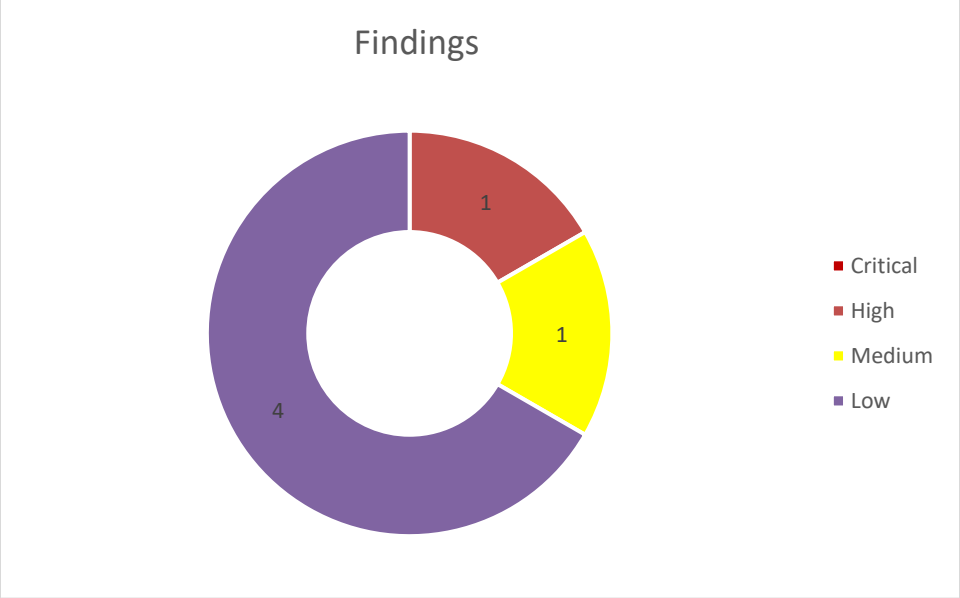

Figure 1. Findings chart

Severity	# of Findings
CRITICAL	0
HIGH	1
MEDIUM	1
LOW	4

2. FINDINGS REPORT

2.1 Critical

Not found

2.2 High

2.2.1 Missing PDA Lifecycle Management (Bet / Round Accounts Never Closed)

Severity: High

Status: Fixed

Location: place_bet_internal(), start_round_internal()

Description

Every time the user places a bet (PlaceBet) or a new round is created (StartRound), the program initializes new PDAs. However, these PDAs are never closed:

- Bet PDA → is created for every bet
- Round PDA → is created every round

The program originally contains no:

$$\#[account(close = user)]$$

$$account.realloc(0)$$

or any custom closure instruction.

Why This Is Important:

Solana PDAs require rent-exempt SOL, which remains locked forever.

Client-reported numbers:

- ~4000 PDAs per week → ≈ 6 SOL wasted on rent
- With realistic traffic 20–40k PDAs/day → $3k–$5k/day rent leakage
- Confirmed on-chain by client's telemetry

This leads to a massive permanent SOL loss.

Problematic Code Examples:

Bet PDA

```
#[account(
    init,
    payer = user,
    space = Bet::LEN,
    seeds = [BET_SEED, round.key().as_ref(), user.key().as_ref()],
    bump
)]
pub bet: Account<'info, Bet>,
```

Round PDA

```
#[account(
    init,
    payer = operator,
    space = Round::LEN,
    seeds = [ROUND_SEED, market.key().as_ref(), &(market.current_epoch + 1).to_le_bytes()],
    bump
)]
pub round: Account<'info, Round>,
```

Fix

1. Close Bet PDA during claim:

```
#[account(
    mut,
    close = user,
    seeds = [BET_SEED, round.key().as_ref(), user.key().as_ref()],
    bump = bet.bump
)]
pub bet: Account<'info, Bet>,
```

2. Round PDA can be closed after all rewards are claimed (via new close_round instruction).

3. Nonce PDA may safely be redesigned into a reusable PDA (optional — low rent impact).

2.3 Medium

2.3.1 Incorrect Logic in Claiming Rewards: Partial Withdrawals Impossible

Severity: Medium

Status: Fixed

Location: claim_treasury_rewards(), claim_referral_rewards()

Description

The incorrect validation used:

$$require!(withdraw\_amount >= withdrawable, PredictionError::InsufficientBalance);$$

This means:

- Cannot withdraw less than the full amount
- Can only withdraw exactly the maximum
- Partial withdrawals are impossible

This breaks protocol operations, since the treasury authority may need partial withdrawals. And withdraw amount must not be more than the withdrawable amount.

Fix

```
- require!(withdraw_amount >= withdrawable,PredictionError::InsufficientBalance);
+ require!(withdraw_amount <= withdrawable, PredictionError::InsufficientBalance);
```

2.3.2 Rent Logic Error: Using wrong minimum rent balance

Severity: Medium

Status: Fixed

Location: claim_treasury_rewards()

Description

The Treasury PDA is created with 8 bytes:

```
#[account(
```

```
    init,
    payer = authority,
    space = 8,
    seeds = [TREASURY_SEED, market.key().as_ref()],
    bump
)]
pub treasury_vault: AccountInfo<'info>,
```

But rent calculation is used incorrectly:

$$let\ rent\_exempt\_minimum\ =\ rent.minimum\_balance(0);$$

This mismatch means the program was computing the rent exemption incorrectly.

Why This Is Important:

- Contract may allow withdrawing too much, leaving PDA non rent-exempt
- This PDA account may be cleaned out by the system, if the balance is lower than needed for rent.

Fix

```
- let rent_exempt_minimum = rent.minimum_balance(0);
+ let rent_exempt_minimum = rent.minimum_balance(treasury_vault.data_len());
```

2.4 Low

2.4.1 Excessive Padding in Bet / Market Increases Rent Costs

Severity: Low

Status: Fixed

Location: state/bet.rs::Bet::LEN, state/market.rs::Market::LEN

Description

Both (Bet and Market) structs use unnecessary padding:

$$+\ 16;\ //\ padding$$

Anchor already aligns fields automatically. Extra padding increases PDA rent cost.

Why This Is Important:

For large-scale traffic (20–40k PDAs/day):

- +16 bytes per PDA
- ≈ 0.0002 SOL extra per account
- 3–6 SOL/day wasted

Fix

Remove padding:

```
- + 16; // padding
+ + 0; // padding removed
```

Use dynamic resizing if needed:

```
account.realloc(new_size, false)?;
```

2.4.2 Redundant Fields in Round Increase Rent Costs

Severity: Low

Status: Fixed

Location: state/round.rs::Round

Description

The Two fields served no logical purpose:

- $reward\_base\_cal\_amount$
- $oracle\_called$

They were neither used nor referenced in any state machine logic. This increased Round PDA size and thus the rent exemption cost.

Why This Is Important

- Every round creates a new PDA
- Larger PDA → higher rent cost
- Unnecessary 17–24 bytes per PDA
- Over ~288 rounds/day → tens of SOL/year wasted

Fix

Remove unused fields:

```
- pub reward_base_cal_amount: u64,
- pub oracle_called: bool,
```

Recompute reward_base on demand:

```
let reward_base = total_amount - treasury_fee_amount;
```

2.4.3 Unreachable Tie-Case Logic in calculate_reward

Severity: Low

Status: Fixed

Location: state/round.rs::Round::calculate_reward()

Description

The reward calculating function contained unreachable logic:

```
if !won { return 0; }
if self.close_price == self.lock_price { ... } // unreachable
// If won == false, we return early, making the tie-case unreachable.
```

Fix

Either remove the function (unused), or correct logic:

```
if self.close_price == self.lock_price {
    return bet_amount * self.reward_base_cal_amount / self.total_amount;
}

let won = match position {
    BetPosition::Bull => self.close_price > self.lock_price,
    BetPosition::Bear => self.close_price < self.lock_price,
};

if !won { return 0; }
```

2.4.4 Field *market.treasury_amount* Is Never Used (Dead & Potentially Dangerous)

Severity: Low

Status: Acknowledged

Location: state/market.rs, execute_round_internal, claim_treasury_rewards, claim_referral_rewards

Description

The market.treasury_amount field in the Market account is updated in several places, but the protocol never actually *uses* it for any payout calculation or withdrawal validation.

Examples:

- The protocol always uses the real PDA balance for withdrawals:
- let actual_balance = treasury_vault.lamports();
- Reward/refund logic never reads market.treasury_amount
- No function validates consistency between treasury_amount and actual PDA lamports

This makes treasury_amount a dead accounting variable — it is written to but never read.

Why the field is dangerous

Even though the variable is currently unused, its existence creates a false sense of trust in an internal accounting system.
This becomes dangerous under two conditions:

1. Future developers may rely on it, assuming it reflects real funds
   (but anyone can transfer SOL directly to a PDA, desynchronizing the field).
2. If later logic is added that reads treasury_amount, it will be incorrect by design, because:
   - third parties can increase/decrease the PDA balance
   - treasury_amount may drift out of sync with the real balance
   - inconsistent accounting opens the door to future payout miscalculations

Why It Is Not a Vulnerability Today

In the current deployment:

- All financial operations rely on the real PDA balance
- treasury_amount is never used in any critical path

Recomendation

Either remove the function (unused), or correct logic:

```
- pub treasury_amount: u64,
```

If accounting is needed, use a single source of truth by calculating it every tme:

```
let rent = Rent::get()?;
let rent_exempt_minimum = rent.minimum_balance(treasury_vault.data_len());
let treasury_amount = treasury_vault.lamports().saturating_sub(rent_exempt_minimum);
market.treasury_amount = treasury_amount;
```

CONCLUSION

The security audit identified several high-priority inefficiencies and logic flaws related to PDA lifecycle management, rent-exempt balance handling, and internal accounting consistency. High-impact issues such as unclosed Bet/Round PDAs and incorrect withdrawal/rent logic were fully addressed, restoring correct lifecycle behavior and ensuring that rent requirements cannot be violated. Medium- and low-severity findings—including unnecessary padding, redundant fields, unreachable code paths, and the unused treasury_amount variable—were also resolved, improving both performance and maintainability. As a result of these fixes, the protocol's architecture became significantly more robust, predictable, and cost-efficient for long-term operation.

REFERENCES