RISK AUDIT

for

BananaZone

on

November 29, 2025

FIDESIUM

Executive Summary

Report

37
/100

TOTAL

Medium risk

November 29, 2025

Abstract

Fidesium's automated risk assessment service was requested to perform a risk posture audit on Banana Protocol **contracts**

Repository Link:
https://github.com/memeworldorg/contracts

Initial Commit Hash:

`b1afcac6ae9a21f2d3ec3c98c47037c351ae9fbe`

Issue Summary

Critical	High	Medium	Low	Info
1 Issues	1 Issues	3 Issues	1 Issues	1 Issues

Caveats

Banana Protocol's codebase is well written, but does incur a handful of high value flaws.

Test Approach

Fidesium performed both Whitebox and Blackbox testing, as per the scope of the engagement, and relied on automated security testing.

Methodology

The assessment methodology covered a range of phases and employed various tools, including but not limited to the following:

- Mapping Content and Functionality of API
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Brute Force Attempt
- Input Handling
- Source Code Review
- Fuzzing of all input parameter
- Dependency Analysis

Severity Definitions

Critical	The issue can cause large economic losses, large-scale data disorder or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defence in Depth.

Risk Issues

Vunerability	Description	Risk	Probability	Status
Missing PDA key equality checks	Multiple functions do not validate PDA key equality before use	Critical	High	Active
Reentrancy	sign_withdraw_proposal modifies state before checking signer	High	Medium	Active
Unsafe cast/truncation	Multiple functions do not check bounds on uint conversion	Medium	Medium	Active
Unsafe unwrap	Multiple functions use unsafe unwraps	Medium	Medium	Active
Missing Slice validation	Serialization without slice length verification	Medium	Medium	Active
Unbounded Vector growth	Possible DoS and Resource exhaustion	Low	Medium	Active
Lamport leakage in PDA Creation	Lamport leakage in PDA Creation	Info	Low	Active

Risk Overview

Team Risk

Low risk: 1

No issues found in founding team

Doxxing Status	Team Experience	Risk Summary
Public	Highly relevant	Low

Liquidity

Risk summary: N/A

As this is a Github assessment, liquidity risks have not been assessed

Whale Concentration

Risk summary: N/A

As this is a Github assessment, whale risks have not been assessed

Smart Contract Risks

Risk summary: 42

The contracts are mostly well written, but have a handful of flaws that should to be carefuly managed.

Vulnerabilities Critical

Missing PDA key equality checks

Vulnerability severity: **Critical**

Vulnerability probability: **High**

Multiple functions do not validate PDA key equality before use

The program derives PDAs but in many places does not assert that the account passed by the client equals the derived PDA. This lets a malicious client pass an arbitrary account (attacker-controlled) and get the program to write sensitive state or receive lamports intended for the PDA.

The following is a list of vulnerable functions

- `create_withdraw_proposal` with PDA: `withdraw_proposal_address`
- `init_withdraw_counter` with PDA: `withdraw_counter_address`
- `init_competition_counter` with PDA: `competition_counter_address`
- `init_config` with PDA: `config_account`
- `create_bet` with PDA: `treasury_ref_account`
- `create_bet` with PDA: `pool_account`
- `create_bet` with PDA: `outcome_range_account`

Recommendations:

Add the following checks directly after deriving the PDAs (modify the derived name to fit the real derived name)

```
if withdraw_proposal_account.key != &withdraw_proposal_address {
    return Err(InvalidWithdrawProposal.into());
}
if withdraw_proposal_account.owner != program_id {
    return Err(InvalidOwner.into());
}
if withdraw_proposal_account.data_len() < (EXPECTED_WITHDRAW_PROPOSAL_LEN) {
    return Err(InvalidAccountSize.into());
}
```

Vulnerabilities High

Reentrancy

Vulnerability severity: **High**

Vulnerability probability: **Medium**

sign_withdraw_proposal modifies state before checking signer

```
fn sign_withdraw_proposal(...) {

    proposal.signers.push(authority.key.to_bytes());
    proposal.signatures_collected += 1;

    if already_signed {
        return Err(AlreadySigned.into());
    }
```

Recommendations:

Move all mutation after signer checks

Vulnerabilities Medium

Unsafe cast/truncatio

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

distribute_payouts does not check bounds on uint conversion

If the division result exceeds u64::MAX, the cast truncates silently. This may produce incorrect payout calculations or cause downstream logic to behave in unsafe ways. In finances, any silent truncation is unacceptable.

```
let winner_bet_index: u64 = outcome_range_state
    .outcome_result_price
    .checked_div(pool_config.volatility as u128)
    .ok_or(ArithmeticError)? as u64;
```

Recommendations:

Use u64::try_from(...) and return a meaningful program error if the value is too large

Unsafe unwrap

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

Multiple functions use unsafe unwraps

try_into() can fail; unwrap() will panic and abort the program with a generic panic code.

Recommendations:

Replace unwrap() with .try_into().map_err(|_| ArithmeticError)? or a dedicated error explaining overflow/invalid allocation size.

Vulnerabilities Medium

Missing Slice validation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

Serialization without slice length verification

Solana runtime does not enforce size correctness. By creating a smaller than expected account and passing this in, you expose the program to DoS.

```
pool_config.serialize(&mut &mut pool_account.data.borrow_mut()[..55])?;
pool_bets_count.serialize(&mut &mut pool_account.data.borrow_mut()[55..59])?;

let record_offset = (bets_checked as usize * 26) + 59;
let record_slice = &mut pool_account.data.borrow_mut()[record_offset .. record_offset+26];
```

try_into() can fail; unwrap() will panic and abort the program with a generic panic code.

Recommendations:

Enforce PDA address correctness, enforce account ownership, and assert data length

```
let (expected_pda, _bump) =
    Pubkey::find_program_address(&[SEED, &pool_id.to_le_bytes()], program_id);

if pool_account.key != &expected_pda {
    msg!("PDA mismatch");
    return Err(ProgramError::InvalidArgument);
}

if pool_account.owner != program_id {
    msg!("Invalid owner for pool account");
    return Err(ProgramError::IllegalOwner);
}

let data = pool_account.data.borrow();
let expected_len = POOL_STATE_LEN;

if data.len() < expected_len {
    msg!("Account is too small: {} < {}", data.len(), expected_len);
    return Err(ProgramError::InvalidAccountData);
}
```

Vulnerabilities Low

Unbounded Vector growth

Vulnerability severity: **Low**

Vulnerability probability: **Medium**

Possible DoS and Resource exhaustion

Multiple vectors (bets, outcomes_ranges, signers) grow without bounds.

```
proposal.signers.push(authority.key.to_bytes());
```

Recommendations:

Enforce maximum vector size limits and validation

Vulnerabilities Info

Lamport leakage in PDA Creation

Vulnerability severity: **Info**

Vulnerability probability: **Low**

Lamport leakage in PDA Creation

```
let value: u64 = **pda.try_borrow_lamports()?;
if pda.owner == &ID && value != 0 {
    invoke_signed(
        &system_instruction::transfer(&pda_address, payer.key, value),
        &[pda.clone(), payer.clone()],
        &[&[seeds, &[bump]]],
    )?;
}
```

If the PDA already exists with lamports, they're transferred to the payer, potentially causing accounting inconsistencies.

Recommendations:

Implement idempotent PDA cration - if it already exists, succeed silently

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer’s risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.