SMART CONTRACTS REVIEW

zokyo

June 6th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING DEFAI REWARDS & BOOSTY

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 1 Medium issue: 1 resolved = 0 points deducted
- 1 Low issue: 1 resolved = 0 points deducted
- 1 Informational issue: 1 acknowledged = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

This document outlines the overall security of the DEFAI REWARDS & Boosty smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the DEFAI REWARDS & Boosty smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Solana network's fast-paced and rapidly changing environment, we recommend that the DEFAI REWARDS & Boosty team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the DEFAI REWARDS & Boosty repository:
Repo: https://github.com/BoostyLabs/eliza-staking-lp-positions/tree/master/programs/eliza_staking/src

Documentation: https://github.com/BoostyLabs/eliza-staking-lp-positions/tree/master/docs

Last commit - 415255ba835459aab1e83ad4efa1f71081ac319b

Contracts under the scope:

- programs/eliza_staking/src
- ├── constants.rs
- ├── error.rs
- ├── instructions
- │ ├── claim_rewards.rs
- │ ├── harvest_position.rs
- │ ├── increase_pool_rewards.rs
- │ ├── init_eliza_config.rs
- │ ├── init_pool.rs
- │ ├── init_stake_entry.rs
- │ ├── mod.rs
- │ ├── stake.rs
- │ ├── unstake.rs
- │ ├── update_eliza_config.rs
- │ └── update_pool_config.rs
- ├── lib.rs
- ├── state
- │ ├── eliza_config.rs
- │ ├── mod.rs
- │ ├── pool_state.rs
- │ └── staked_position.rs
- └── utils.rs

Documentation:

- docs
- ├── anchor_rules.md
- ├── eliza_staking_features.md
- ├── instructions
- │ ├── claim_rewards.md
- │ ├── harvest_position.md
- │ ├── increase_pool_rewards.md
- │ ├── initialize_eliza_config.md
- │ ├── initialize_pool.md
- │ ├── initialize_stake_entry.md
- │ ├── stake_flow.md
- │ ├── stake.md
- │ ├── unstake.md
- │ ├── update_eliza_config.md
- │ └── update_pool_config.md
- ├── reward_calculation.md
- ├── scripts
- │ ├── orca_scripts.md
- │ ├── rewards-calculator.md
- │ ├── script_execution_guide.md
- │ └── scripts.md
- ├── Solana_LP_Eliza_Staking_PRD.md
- └── Technical_Architecture.md

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of DEFAI REWARDS & Boosty smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contract/s by industry leaders.
03	Testing contract/s logic against common and uncommon attack vectors.

Executive Summary

All findings were validated with reference to exact logic in the program files and cross-verified against Anchor account constraints and CPI behavior. No critical vulnerabilities (e.g., fund-stealing, account corruption, authority bypass) were identified. The program demonstrates robust architectural patterns, safe instruction ordering, and good use of Anchor account validation macros.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the DEFAI REWARDS & Boosty team and the DEFAI REWARDS & Boosty team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Exponential Time Curve Overflow Risk	Medium	Resolved
2	No update_rewards() in update_pool_config	Low	Resolved
3	Single-Signer Authority Introduces Centralization Risk	Informational	Acknowledged

Exponential Time Curve Overflow Risk

The `stake()` instruction calculates the reward multiplier for exponential time curves as:

```
let yearly_factor = (exponent - FP_SCALE) * days / 365;
let multiplier = base * (FP_SCALE + yearly_factor) / FP_SCALE;
```

This uses saturating math `(saturating_mul, saturating_div)` but all variables are `u64`, so extreme values like:

- `stake_duration` of thousands of years,
- or `exponent` values >> `FP_SCALE`

...can silently result in incorrect multipliers, reward miscalculations, or reward clipping due to silent overflow. There's currently no cap on `stake_duration`.

Recommendation:

- Cap `stake_duration` to a reasonable maximum (e.g. 10 years or 3650 days).
- Add unit tests to simulate high-duration + high-exponent scenarios and verify that saturation behaves as intended.

No `update_rewards()` in `update_pool_config`

The `update_pool_config` instruction allows the pool authority to change reward-affecting parameters like:

- `reward_duration`
- `curve_slope_or_exponent`
- `is_curve_linear`

...but does not call `update_rewards()` before modifying them. While this does not cause reward loss or misaccounting (because user flows call `update_rewards()` anyway), it may cause temporary timing drifts in the pool-level snapshot of rewards.

Recommendation:

- Call `update_rewards()` at the start of `update_pool_config` when reward-affecting parameters are modified.
- This ensures clean alignment of the pool's reward accrual logic and avoids reward drift windows.

Single-Signer Authority Introduces Centralization Risk

Critical administrative instructions such as `update_pool_config` and `increase_pool_rewards` require only a single authority (`Signer`) to execute. If this key is compromised, the attacker can:

- Change reward settings
- Drain or misroute rewards
- Pause/unpause pools
- Reassign control

This introduces a central point of failure in an otherwise decentralized protocol.

Recommendation:

- Use a multisig (e.g. SPL Multisig, Squads) or governance DAO program to manage pool and protocol authorities.
- Alternatively, separate operational keys from configuration keys to reduce attack surface.

	Contracts
Re-entrancy	Pass
Access Control / Role Hierarchy	Pass
Arithmetic Overflows / Underflows	Pass
Unexpected SOL / Native Token Transfers	Pass
Unsafe Cross-Program Invocation (CPI)	Pass
Default Visibility of Structs / Functions	Pass
Hidden Malicious Logic (e.g., in Traits or Unsafe Blocks)	Pass
Entropy Illusion / Weak Randomness (e.g., using clock.unix_timestamp)	Pass
External Program Calls (CPI) Trust Model	Pass
Short Account Address Attack (especially in Solana instruction deserialization)	Pass
Unchecked Return Values from CPIs	Pass
Race Conditions / Front-Running in Account Loading	Pass
Denial of Service (DOS) via Compute Budget Exhaustion	Pass
Uninitialized Accounts or Fields	Pass
Precision / Floating Point Misuse (use fixed-point where needed)	Pass
Cross-Program signer seeds Mismanagement	Pass
Signature Replay (e.g., reused instructions or approvals)	Pass
Backdoors in Token Programs / Custom SPL Implementations	Pass
Improper Deserialization / Manual Offset Handling Bugs	Pass
Improper Use of unsafe Rust Blocks	Pass
Missing or Misused Seeds in Pubkey::find_program_address()	Pass
Improper PDA Authority Handling	Pass
Lack of Rent Exemption Checks	Pass
Unchecked Account Ownership / is_signer Flags	Pass
Time-Based Logic Vulnerabilities (e.g., timestamp races)	Pass

We are grateful for the opportunity to work with the DEFAI REWARDS & Boosty team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the DEFAI REWARDS & Boosty team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo