Armada Program Suite Audit Report

06.02.2024

– Mad Shield

Contents

01- Introduction

ArmadaFi engaged Mad Shield to audit two programs including the `spl-token-staking` and the `clp-vault` program. The audit included a full analysis of the core functionality of both programs, with subsequent reviews following major releases. We performed multiple intermittent cycles of intense security analysis of the codebase to ensure the highest safety standards.

Our team was well aware of the original algorithm of the `spl-token-staking` program. The re-implementation of the program in the Anchor framework is very well-written, clear and concise which improves significantly upon the first implementation. The Mad Shield team is glad to announce that we discovered **0 vulnerabilities** in the `spl-token-staking` program.

For the `clp-vault` program, we analyzed the program's security measures over the time period between November 2023 to January 2024. In summary, we uncovered a total of **2 vulnerabilities**; 1 Medium and 1 Low-severity vulnerability.

A detailed walk-through of our findings is provided in this report. We communicated all our findings with proper mitigations through our private channels to the ArmadaFi development team. We are glad to confirm that all vulnerabilities reported during the audit have been addressed and resolved accordingly. This report documents the audit process, the assessment methodology, and declares this program to be secure to the best of our knowledge.

1.1 Overview

ArmadaFi provides end-to-end tooling to power the Solana SPL token ecosystem. It is fully customizable to support demands tailored to the needs of all solana-based projects.

This audit covers the security for two of the main programs that underlie the Armada's fleet of products.

- **SPL Token Staking –** A novel solution to allow users to easily setup and interact with tokenized stake and reward pools for use in SPL governance without making unnecessary compromises between voting power and collected rewards.
- **Concentrated Liquidity Pool (CLP) vaults –** A solution built to allow users to create and manage market making positions (MM) in a specific Whirlpool, where 2 assets trade against each other.

This report presents a comprehensive analysis of the implemented changes and their security implications. The main purpose is to identify and remedy program vulnerabilities.

1.2 System Specification

This section covers the technical intricacies of the CLP vaults for market making management and the `spl-token-staking` program for DAO governance. This section is not intended to serve as documentation for these programs. Instead, it highlights specific features which are most relevant to the audit. For official documentation, please visit https://docs.armadafi.so/.

1. **Position Management (CLP Vaults)**

   ArmadaFi enables instant trading within ecosystems through the use of concentrated liquidity pool (CLP) vaults which automate the process of on-chain liquidity provisioning for SPL tokens. These CLP vaults are built upon industry-leading practices providing enterprise-grade market maker management.

   Each `clp_vault` is built with a different strategy based on the goal of the vault. The ranges for each position are managed by the vault, removing the requirement of the LP to set its own ranges.The users deposit assets into any of the vaults without a lockup requirement where each vault supports up to 5 open positions at any given time. The `clp_vault` program provides important interactive functions including `increase_liquidity`, `open_position` and `withdraw` to manage the CLP vaults.

   The vaults are built on top of concentrated liquid market making (CLMM) pools. The pools utilize Orca Whirlpool configs, which are double audited and open-source. The Orca Whirlpool configs are available at https://github.com/orca-so/whirlpools.

2. **Governance and Tokenomics (SPL Token Staking)**

   ArmadaFi `spl-token-staking` program provides tokenized representation of staked tokens to be used for SPL Governance. This is a revolutionary development that solves the user dilemma of picking one of the two apparent benefits; choosing to transfer tokens to SPL Governance (to vote on DAO proposals) or stake tokens to receive rewards.

   The program minimizes the amount of account types (account efficient) needed to support complex staking activities. Specifically, it only has 3 account types which can accept many tokens as both stake and rewards tokens.

The first account type is a `StakePool` which must be initialized using a `creator` wallet to use the `spl-token-staking` program. This wallet may serve as an `authority` to make updates to StakePool. It also keeps track of `total_weighted_stake` (total amount of staked tokens affecting lock up period weighting), `stake_mint` (address representing effective stake), `reward_pools` (RewardPools applicable to the stake pool) and variables relating to the max/min weight (for staking) and duration for lockup. The staking weight is scaled linearly from the minimum to the maximum weight with respect to time.

Compared to the first audit, the new program provides the ability for governance to the stakers. This is enabled through the `stake_mint` which will be minted to the stakers according to the total weight of their stake representing their share of the pool. With thorough analysis, the Mad Shield team made sure that this function is secure and does not exhibit any malicious behavior to the rest of the program's functionality; preventing unexpected or malicious transfer of funds out of the pool. It is important for the users to be wary of transferring/losing their tokens as it will nullify their ability to withdraw their stake.

After initialization, a `RewardPool` can be added using the authority address used to initialize StakePool. Each StakePool can initialize up to 10 `RewardPools`, supporting up to 10 different SPL tokens as staking rewards. Rewards are distributed based on the weight of the stake at any given time. The reward pool has 3 important members: `reward_vault` (token account of the SPL Token reward), `rewards_per_effective_stake` (the amount every staked token earns from the vault rewards), and `latest_amount` (latest amount of tokens in vault).

The `StakeDepositReceipt` account is created when you stake an asset. It keeps track of information related to the stake including the `owner` (who owns the staked assets), `payer` (address paying for deposit), address of the `stake_pool`, `deposit_amount`, `lockup_duration`, `effective_stake` (amount of stake weighted by lockup), etc.

Fig.1 provides an overview of the account structure and program flow of the `spl-token-staking` program.

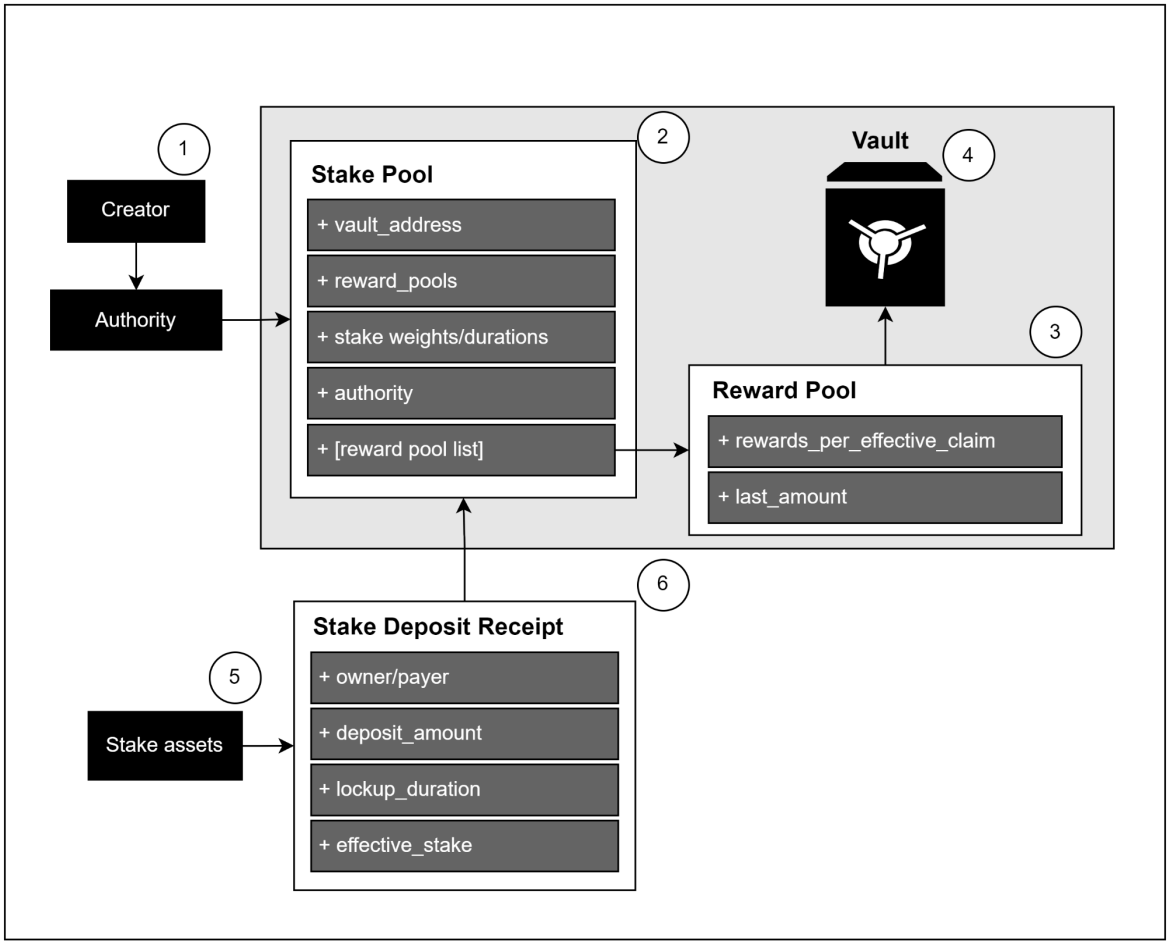

Fig 1: Program structure and account interactions for `spl-token-staking`

02. Scopes and Objectives

The main objectives of the audit are defined as:

- Minimizing the possible presence of critical vulnerabilities in the program. This would include detailed examination of the code and edge case scrutinization to find as many vulnerabilities.
- 2-way communication during the audit process. This included for Mad Shield to reach a perfect understanding of the design of `spl-token-staking` and `clp-vault` programs and the goals of the team.
- Provide clear and thorough explanations of all vulnerabilities discovered during the process with potential suggestions and recommendations for fixes and code improvements.
- Clear attention to the documentation of vulnerabilities with the eventual publication of a comprehensive audit report to the public audience for all stakeholders to understand the security state of the programs.

ArmadaFi has delivered these programs to Mad Shield at the following Github repositories.

1. SPL Token Staking

Repository URL	https://github.com/mithraiclabs/spl-token-staking
Commit (start of audit)	b65c639aa1a30770018b169112d024d74ed01ed4
Commit (end of audit)	9c2ee436885cd28f5becc3c9c789438f7e4d37c0

The scope encompasses **pull requests** [#15, #16, #17] from the `spl-token-staking` program.

2. CLP-Vault

Repository URL	https://github.com/mithraiclabs/psylbc (*private*)
Commit (start of audit)	c4284db173e6993e021f06345bd37014960df683
Commit (end of audit)	b99342190ee1845144619d584c6fa7691c7527c8

The scope encompasses **pull requests** [#333, #362, #364] from the `clp-vault` program.

03 . Methodology

After initial contact with the ArmadaFi team, we did a quick analysis of the source code to evaluate the scope of the work and recognize potential footguns. Due to the complex nature of the staking algorithms, we were in constant contact with the ArmadaFi development team to realize the specification details about the program. After realizing the scope of the project, we began the in-depth examination of the codebase.

As per Mad Shield promise, our team ran intense fuzzy and penetration tests, hitting the program with custom transactions with randomly generated data and different account types to uncover any residual vulnerabilities that might put the program in danger. The program did not exhibit any unexpected or erroneous executions.

Next we shifted focus to the `clp-vault` program. We analyzed the correctness and safety of the program with regards to composability with other programs. `clp-vault` communicates with Whirlpool, an external program interface to achieve desired functionality. We did an in-depth read-through of the Whirlpool protocol to ensure the boundaries of the program interactions are safe.

Mad Shield overall discovered **2 vulnerabilities** within the `clp-vault` program. We uncovered the first by going through the testing framework of Armada; there is a significant amount of testing for the `clp-vault`. By utilizing this extensive testing environment, we found a bug concerning the instruction introspection for the external swaps as outlined in **Shield_ARM_01** in the findings and recommendations section.

As per our responsibilities, we made sure all specifications of the program are well understood and work as intended by extensively engaging the development team at ArmadaFi. The second bug was discovered through this involvement with the team to nail down the specification. This vulnerability is detailed in **Shield_ARM_02.**

04 . Findings & Recommendations

Our severity classification system adheres to the criteria outlined here.

Severity Level	Exploitability	Potential Impact	Examples
Critical	Low to moderate difficulty, 3rd-party attacker	Irreparable financial harm	Direct theft of funds, permanent freezing of tokens/NFTs
High	High difficulty, external attacker or specific user interactions	Recoverable financial harm	Temporary freezing of assets
Medium	Unexpected behavior, potential for misuse	Limited to no financial harm, non-critical disruption	Escalation of non-sensitive privilege, program malfunctions, inefficient execution
Low	Implementation variance, uncommon scenarios	Zero financial implications, minor inconvenience	Program crashes in rare situations, parameter adjustments by authorized entities
Informational	N/A	Recommendations for improvement	Design enhancements, best practices, usability suggestions

In the following, we enumerate some of the findings and issues we discovered and explain their implications and corresponding resolutions.

Finding	Description	Severity Level
`clp-vault`		
SHIELD_CLP_01 [RESOLVED]	Malicious market maker behavior possible during the swap transaction, marking market maker semi-trusted.	*Medium*
SHIELD_CLP_02 [RESOLVED]	A new depositor may overpay for LP mint	*Low*
`spl-token-staking`		
SHIELD_SLP_GR_01	Unchecked arithmetic operations.	*Informational*
SHIELD_SLP_GR_02	Non-deterministic nonce for PDA seeds.	*Informational*

1. **Shield_ARM_01 - Malicious market maker behavior possible during the swap transaction [Medium]**

   The current implementation assumes the MM is semi-trusted as there is a possibility of malicious behavior during a swap transaction. However, the program makes extra effort to minimize the trust in the market maker as much as possible.

   The function `ext_swap_setup` allows for a market maker to swap the reserve assets on a 3rd party exchange such as Jupiter to allow more efficient position rebalancing. To this end, the market maker is approved as a token account delegate in an `exp_swap_setup` instruction. This authority is subsequently revoked in an `exp_swap_cleanup` instruction.. The program enforces the presence of both these instructions through instruction introspection.

   However, this check comes short as `exp_swap_setup` can be called again for a second `clp_vault` in between the `exp_swap_setup` and `exp_swap_cleanup` flow for the initial `clp_vault` in the instruction. This essentially approves the MM to spend all tokens from the second `clp_vault` reserves. This happens as the program does not perform thorough instruction introspection and returns successfully even though no corresponding `exp_swap_cleanup` instruction is present to revoke the approval from the second call.

   This enables the market maker to deplete the reserves in a following TX essentially having explicit access to the funds in a very simple attack requiring no sophisticated instruction manipulation.

   Recommended Fix

   To fix the issue, we suggested that the program checks that the setup/cleanup instructions are called exactly once, which will further minimize the trust assumptions in the Market Maker.

   The patch commit: [b9934219]

2. Shield_ARM_02 - A new depositor may underpay for a LP mint [Low]

Each LP-mint token represents a share of the assets that belong to the vault across all 3 of

a. Liquidity that has been deposited into the whirlpool (position accounts)
b. Liquidity available in the reserves that has not been deployed
c. Uncollected fees and rewards from whirlpool

The Armada protocol also takes a performance fee for providing this infrastructure. Therefore, the total value that new depositor has to deposit for lp_mint amount of tokens is

$$deposit\_amount = (reserve + (pool\_fees \times (1 - armada\_fee))) \times \frac{lp\_mint}{supply}$$

For this calculation to be correct, the fee summation must occur before the multiplication by the mint LP-ratio. However, the program would multiply and divide the accrued fee for each vault position before the final summation. As such, the `deposit_amount` would sum up to a smaller amount as each term would suffer from a loss in precision.

```
// CURRENT: BAD
deposit_amount = (reserve +
  (fee_1 * lp_mint / supply + fee_2 * lp_mint / supply + ... )
);

// SUGGESTED: GOOD
deposit_amount = reserve  + (fee_1 + fee_2 + ... ) * lp_mint / supply;
```

Recommended Fix

The suggested calculation from deposit amount will keep the `lp_mint` tokens from fluctuating in value across deposits. This also prevents new depositors from underpaying for tokens where the offset loss gets distributed amongst the current depositors.

The patch commit: a8b2175e

05. General Recommendations

In this section, we provide general recommendations and informal issues that we found during the process of the audit.

1. **Shield_ARM_GR_01 - Unchecked arithmetic operations**

   There are occurrences of vanilla Rust additions and divisions in the source code. For example, the following code snippet calculates the mid-duration for a stake time period. We suggest always using the checked operations for extra protection.

```rust
// file: state.rs
fn get_stake_weight_duration_midpoint () {
    // ..
    let mid_duration = (min_duration + max_duration) / 2;
}
```

2. **Shield_ARM_GR_02 - Non-deterministic nonce for PDA seeds**

   The `StakeDepositReceipt` account created by user deposit instruction has the following seeds:

```rust
// file: deposit.rs
seeds = [
  owner.key().as_ref(),
  stake_pool.key().as_ref(),
  &nonce.to_le_bytes(),
  b"stakeDepositReceipt",
]
```

It is possible that reused nonce could invalidate the indexer data, resulting in invalid front-end statistics. As per our recommendation, the team has remedied this through keeping a deterministic nonce for the `StakeDepositReceipt` seeds.

Conclusion

In conclusion, Mad Shield's audit of the `spl-token-staking` and `clp-vault` programs has been a thorough effort to enhance security. This is the second of the two audits conducted by the Mad Shield team. The bugs from the previous version have been patched so we're pleased to report that there are no vulnerabilities affecting the reliability and security of the program.

The audit underscores our focus on security within the expanding Solana DeFi landscape, emphasizing the ongoing need for security, penetration testing and transparent documentation. We would like to acknowledge the ArmadaFi team for their cooperative approach throughout the process, which has contributed to the overall effectiveness of this collaboration.