SECURITY AUDIT REPORT

for

Orama Launchpad

Prepared By: Xiaomi Huang

PeckShield

September 29, 2025

Document Properties

Client	Orama Labs
Title	Security Audit Report
Target	Orama Launchpad
Version	1.0
Author	Xuxian Jiang
Auditors	Matthew Jiang, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	September 29, 2025	Xuxian Jiang	Final Release
1.0-rc1	September 20, 2025	Xuxian Jiang	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the Orama Launchpad protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Orama Launchpad

Orama Launchpad is committed to building next-generation tokenized asset protocol, aiming to solve key pain points in traditional scientific research—inefficient funding allocation and resource distribution. The platform establishes a closed-loop ecosystem from research to commercialization by funding scientific experiments, enabling intellectual property rights verification, resolving data silos, and implementing community governance, thereby driving innovation in on-chain research. The basic information of audited contracts is as follows:

Table 1.1: Basic Information of Orama Launchpad

Item	Description
Name	Orama Labs
Type	Solana
Language	Rust
Audit Method	Whitebox
Latest Audit Report	September 29, 2025

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/OramaLabs/launchpad-program.git (7d27638)

And here is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/OramaLabs/launchpad-program.git (d091d85)

1.2 About PeckShield

PeckShield Inc. [9] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [8]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact, and can be accordingly classified into four categories, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.

- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [7], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the `Orama Launchpad` implementations. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	4	■■■■
Low	4	■■■■
Informational	0	
Total	8	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 4 medium-severity vulnerabilities and 4 low-severity vulnerabilities.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Medium	Incorrect token_vault Authority in UnstakeToken Struct	Business Logic	Resolved
PVE-002	Low	Strengthened launch_pool Account Validation in Multiple Instruction	Coding Practices	Resolved
PVE-003	Medium	Possible Migration Denial-of-Service With (External) Early DAMMv2 Pool Initialization	Business Logic	Resolved
PVE-004	Medium	Arithmetic Overflow Avoidance in Staking Position Initialization	Business Logic	Resolved
PVE-005	Low	Suggested token_x_program Validation in handle_dlmm_swap()	Business Logic	Resolved
PVE-006	Low	Inconsistent Account Ordering From Meteora DLMM Swap2 IDL	Coding Practices	Resolved
PVE-007	Low	Improper Refunded Flag Update Upon Successful User Claims	Business Logic	Resolved
PVE-008	Medium	Trust Issue of Admin Keys	Security Features	Mitigated

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Incorrect token_vault Authority in UnstakeToken Struct

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `StakeTokens`
- Category: Business Logic [6]
- CWE subcategory: CWE-837 [3]

Description

The `Orama Launchpad` program allows users to stake/unstake their funds to earn rewards. While reviewing the unstaking logic, we notice an issue that does not properly set up the authority for the given `token_vault` account.

In the following, we show the `Anchor`-based accounts validation struct that defines all the accounts needed when a user unstakes tokens from the program. It comes to our attention that one specific account, i,e., `token_vault`, is a `PDA`-owned `SPL` token account. This token account holds the staked tokens and its authority is currently set as `vault_authority` (line 51), which should be `staking_position` instead.

```
9  #[derive(Accounts)]
10 #[instruction(amount: u64, lock_duration: i64)]
11 pub struct StakeTokens<'info> {
12     /// User who wants to stake tokens
13     #[account(mut)]
14     pub user: Signer<'info>,
15 
16     /// vault authority
17     #[account(
18         mut,
19         seeds = [
20             VAULT_AUTHORITY.as_ref(),
21         ],
22         bump,
23     )]
```

```
    pub vault_authority: SystemAccount<'info>,

    /// Global configuration account
    #[account(
        seeds = [GlobalConfig::SEED],
        bump = global_config.bump,
    )]
    pub global_config: Account<'info, GlobalConfig>,

    /// Token mint of the token to be staked
    pub token_mint: Account<'info, Mint>,

    /// User's token account (source of tokens)
    #[account(
        mut,
        token::mint = token_mint,
        token::authority = user,
    )]
    pub user_token_account: Box<Account<'info, TokenAccount>>,

    /// Program's token vault to hold staked tokens
    #[account(
        init_if_needed,
        payer = user,
        seeds = [TOKEN_VAULT, vault_authority.key().as_ref(), token_mint.key().as_ref()],
        bump,
        token::mint = token_mint,
        token::authority = vault_authority,
    )]
    pub token_vault: Box<Account<'info, TokenAccount>>,

    /// Staking position account for this user and token
    #[account(
        init_if_needed,
        payer = user,
        space = StakingPosition::SIZE,
        seeds = [
            StakingPosition::SEED,
            user.key().as_ref(),
            token_mint.key().as_ref()
        ],
        bump,
    )]
    pub staking_position: Box<Account<'info, StakingPosition>>,

    /// Token program
    pub token_program: Program<'info, Token>,

    /// System program
    pub system_program: Program<'info, System>,

```

```
    /// Rent sysvar
    pub rent: Sysvar<'info, Rent>,
}
```

Listing 3.1: The `StakeTokens` Struct

Recommendation Revisit the above `StakeTokens` struct to properly validate the `token_vault` account.

Status The issue has been fixed in the following commit: a591910.

3.2 Strengthened launch_pool Account Validation in Multiple Instruction

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `Multiple Structs`
- Category: Business Logic [6]
- CWE subcategory: CWE-837 [3]

Description

A number of instructions in `Orama Launchpad` program have a common account, i.e., `launch_pool`, which denotes the pool account for a distinct token launch. This pool account itself is a `PDA` derived from `[LAUNCH_POOL_SEED, creator.key(), global_config.pool_count]` and is used to store pool-specific configuration, bump, creator, etc. Our analysis shows that its use can be better validated.

In the following, we show the `Anchor`-based accounts validation struct `ParticipateWithPoints` that represents a user participation on the token launch. This struct defines the `launch_pool` account with a single constraint `constraint = launch_pool.is_active()` (line 44), which can be much strengthened by additionally including the seeds and associated bump for its address validation. Note the same suggestion is applicable to a number of instruction-related structs, e.g., `FinalizeLaunch`, `DammV2`, `ClaimCreatorTokens`, `ClaimUserRewards`, etc.

```
14 #[derive(Accounts)]
15 #[instruction(points_to_use: u64, total_points: u64)]
16 pub struct ParticipateWithPoints<'info> {
17     #[account(mut)]
18     pub user: Signer<'info>,
19 
20     /// CHECK: vault authority
21     #[account(
22         mut,
23         seeds = [VAULT_AUTHORITY.as_ref()],
```

```
        bump,
    )]
    pub vault_authority: SystemAccount<'info>,

    /// Global configuration account
    #[account(
        seeds = [GLOBAL_CONFIG_SEED],
        bump = global_config.bump,
    )]
    pub global_config: Box<Account<'info, GlobalConfig>>,

    /// CHECK: WSOL mint (verified by address)
    #[account(
        address = anchor_spl::token::spl_token::native_mint::ID
    )]
    pub wsol_mint: Account<'info, Mint>,

    /// Launch pool account
    #[account(
        mut,
        constraint = launch_pool.is_active() @ LaunchpadError::LaunchNotActive,
    )]
    pub launch_pool: Box<Account<'info, LaunchPool>>,

    /// User points account
    #[account(
        init_if_needed,
        payer = user,
        space = UserPoint::SIZE,
        seeds = [USER_POINT_SEED, user.key().as_ref()],
        bump,
    )]
    pub user_point: Box<Account<'info, UserPoint>>,

    /// User position account
    #[account(
        init_if_needed,
        payer = user,
        space = UserPosition::SIZE,
        seeds = [USER_POSITION_SEED, launch_pool.key().as_ref(), user.key().as_ref()],
        bump,
    )]
    pub user_position: Box<Account<'info, UserPosition>>,

    /// Launch pool WSOL vault (for storing raised SOL)
    /// CHECK: PDA account only for storing SOL
    #[account(
        mut,
        seeds = [TOKEN_VAULT, vault_authority.key().as_ref(), wsol_mint.key().as_ref()],
        bump,
        token::mint = wsol_mint,
        token::authority = vault_authority,
```

```
        token::token_program = token_program
    )]
    pub wsol_vault: Account<'info, TokenAccount>,

    /// System variables account for Ed25519 signature verification
    /// CHECK: This is a system-provided instruction system variable
    #[account(address = sysvar::instructions::ID)]
    pub instructions_sysvar: UncheckedAccount<'info>,

    /// Token program
    pub token_program: Program<'info, Token>,
    pub system_program: Program<'info, System>,
}
```

Listing 3.2: The `ParticipateWithPoints` Struct

Recommendation Revise the above-mentioned structs to strengthen the `launch_pool` validation.

Status The issue has been fixed in the following commit: a591910.

3.3 Possible Migration Denial-of-Service With (External) Early DAMMv2 Pool Initialization

- ID: PVE-003
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `DammV2`
- Category: Business Logic [6]
- CWE subcategory: CWE-837 [3]

Description

The new token launch has five status, i.e., `Initialized`, `Active`, `Success`, `Failed`, and `Migrated`. A successful launch will migrate the liquidity to a `Meteora` pool. While examining the migration logic, we notice a possible issue that may block the `Migrated` status from being entered.

In the following, we show the implementation of the related routine, i.e., `create_pool()`. This routine will be invoked once sufficient funds are raised and the new token launch will migrate the raised liquidity to a `Meteora` pool. However, current implementation makes an implicit assumption that this `Meteora` pool is not created yet. In other words, if the target `Meteora` pool is externally created (even without any liquidity), current execution of `create_pool()` will be reverted.

Moreover, current implementation allows anyone to invoke it with flexibility in choosing the initial price `price`. Since the raised liquidity will be added into the `Meteora` pool, if the pool is initialized in a manipulated initial price or an imbalanced pool, the liquidity addition may create arbitrage opportunity that should be avoided as much as possible.

```
    pub fn create_pool(&mut self, sqrt_price: u128) -> Result<()> {
        let launch_pool = &mut self.launch_pool;

        // Verify launch pool is in correct state
        require!(
            launch_pool.status == LaunchStatus::Success,
            LaunchpadError::InvalidLaunchStatus
        );

        // Verify we have sufficient liquidity to create pool
        require!(
            launch_pool.liquidity_allocation > 0 && launch_pool.liquidity_sol > 0,
            LaunchpadError::InsufficientLiquidity
        );

        let signer_seeds: &[&[&[u8]]] = &[&[VAULT_AUTHORITY, &[VAULT_BUMP]]];

        let config = self.pool_config.load()?;
        let base_amount: u64 = launch_pool.liquidity_allocation;
        let quote_amount: u64 = launch_pool.liquidity_sol;

        let liquidity = get_liquidity_for_adding_liquidity(
            base_amount,
            quote_amount,
            sqrt_price,
            config.sqrt_min_price,
            config.sqrt_max_price,
        )?;

        cp_amm::cpi::initialize_pool(
            CpiContext::new_with_signer(
                self.amm_program.to_account_info(),
                cp_amm::cpi::accounts::InitializePoolCtx {
                    creator: self.vault_authority.to_account_info(),
                    position_nft_mint: self.position_nft_mint.to_account_info(),
                    position_nft_account: self.position_nft_account.to_account_info(),
                    payer: self.vault_authority.to_account_info(),
                    config: self.pool_config.to_account_info(),
                    pool_authority: self.damm_pool_authority.to_account_info(),
                    pool: self.pool.to_account_info(),
                    position: self.position.to_account_info(),
                    token_a_mint: self.base_mint.to_account_info(),
                    token_b_mint: self.quote_mint.to_account_info(),
                    token_a_vault: self.token_a_vault.to_account_info(),
                    token_b_vault: self.token_b_vault.to_account_info(),
                    payer_token_a: self.token_vault.to_account_info(),
                    payer_token_b: self.wsol_vault.to_account_info(),
                    token_a_program: self.token_base_program.to_account_info(),
                    token_b_program: self.token_quote_program.to_account_info(),
                    token_2022_program: self.token_2022_program.to_account_info(),
                    system_program: self.system_program.to_account_info(),
                    event_authority: self.damm_event_authority.to_account_info(),
```

```
                program: self.amm_program.to_account_info(),
            },
            signer_seeds,
        ),
        cp_amm::InitializePoolParameters {
            liquidity,
            sqrt_price,
            activation_point: None,
        },
    )?;
    ...
    let clock = Clock::get()?;
    launch_pool.creator_unlock_start_time = clock.unix_timestamp;

    launch_pool.status = LaunchStatus::Migrated;

    msg!("Creator token unlock will start at: {}", clock.unix_timestamp);
    msg!("Lock duration: {} days", launch_pool.creator_lock_duration / (24 * 3600));
    msg!("Linear unlock duration: {} days", launch_pool.
        creator_linear_unlock_duration / (24 * 3600));

    Ok(())
}
```

Listing 3.3: `DammV2::create_pool()`

Recommendation Revise the above-mentioned routine to ensure the migrate status can be successfully entered with intended pool state.

Status The issue has been fixed in the following commit: 5c6cd51.

3.4 Arithmetic Overflow Avoidance in Staking Position Initialization

- ID: PVE-004
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `StakingPosition`
- Category: Business Logic [6]
- CWE subcategory: CWE-837 [3]

Description

A user may choose to stake in `Orama Launchpad` for potential rewards. The user stake position is initialized or updated once a new stake operation occurs. In the process of examining the staking logic, we notice current implementation can be improved.

```
49    pub fn initialize(
50        &mut self,
51        user: Pubkey,
52        token_mint: Pubkey,
53        staked_amount: u64,
54        lock_duration: i64,
55        current_time: i64,
56        bump: u8,
57    ) {
58        self.user = user;
59        self.token_mint = token_mint;
60        self.staked_amount = staked_amount;
61        self.lock_duration = lock_duration;
62        self.stake_time = current_time;
63        self.unlock_time = current_time + lock_duration;
64        self.bump = bump;
65        self.reserved = [0; 8];
66    }
```

Listing 3.4: `StakingPosition::initialize()`

To elaborate, we show above the implementation of the related routine, i.e., `initialize()`. While the purpose is to initialize the user stake position for the first time, we find it necessary to support multiple stakes, i.e., the staked amount can be updated as `self.staked_amount = self.staked_amount .checked_add(staked_amount)`, not current `self.staked_amount = staked_amount` (line 60). Also, the unlock time may need to be revised as `self.unlock_time = current_time.checked_add(lock_duration)`, not current `self.unlock_time = current_time + lock_duration` (line 63).

Recommendation Revisit the above routine to properly update the user stake position.

Status The issue has been fixed in the following commit: a591910.

3.5 Suggested token_x_program Validation in handle_dlmm_swap()

- ID: PVE-005
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `DlmmSwap`
- Category: Business Logic [6]
- CWE subcategory: CWE-837 [3]

Description

To facilitate the user operations, the `Orama Launchpad` program also provides the support of token swaps via external integration of `DLMM` swap engine. We notice certain fee `0.05%` may be deducted from the input amount. While the fee is deducted and transferred from the user account, we notice the transfer-related `CPI` call invokes an external program `token_x_program`, which needs to be better validated.

```
pub fn handle_dlmm_swap<'a, 'b, 'c, 'info>(
    ctx: Context<'a, 'b, 'c, 'info, DlmmSwap<'info>>,
    amount_in: u64,
    min_amount_out: u64,
    remaining_accounts_info: RemainingAccountsInfo
) -> Result<()> {
    // Calculate 0.05% fee from input tokens (5 basis points)
    let fee_amount = amount_in
        .checked_mul(5)
        .and_then(v v.checked_div(10000))
        .ok_or(ProgramError::ArithmeticOverflow)?;

    // Calculate actual amount to swap after deducting fee
    let actual_swap_amount = amount_in
        .checked_sub(fee_amount)
        .ok_or(ProgramError::ArithmeticOverflow)?;

    // Transfer fee from user's input token account to admin fee account
    if fee_amount > 0 {
        // Use token_x_program since user_token_in is always WSOL
        token::transfer(
            CpiContext::new(
                ctx.accounts.token_x_program.to_account_info(),
                Transfer {
                    from: ctx.accounts.user_token_in.to_account_info(),
                    to: ctx.accounts.admin_fee_token_in.to_account_info(),
                    authority: ctx.accounts.user.to_account_info(),
                },
            ),
            fee_amount,
        )?;
    }
    ...
}
```

Listing 3.5: `DlmmSwap::handle_dlmm_swap()`

To elaborate, we show above the code snippet of the related routine, i.e., `handle_dlmm_swap()`. Specifically, the code snippet is used for fee deduction from the user input token. And the `token::transfer()` call is given a program `token_x_program`, which is currently not validated yet.

Recommendation Revisit the above routine to properly validate the invoked `token_x_program` program.

Status The issue has been fixed in the following commit: a591910.

3.6 Inconsistent Account Ordering From Meteora DLMM Swap2 IDL

- ID: PVE-006
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: DlmmSwap
- Category: Coding Practices [5]
- CWE subcategory: CWE-1126 [1]

Description

When invoking a Cross-Program Invocation (CPI) in Solana, it is normally required to ensure that the instruction's account metas and the associated account_infos array are consistent and correctly ordered. In the process of examining the DLMM Swap call, we notice the ordering of given accounts may not be consistent with the defined IDL from the DLMM Swap program.

To elaborate, we use the accounts struct defined for the invocation of DLMM Swap call. Our analysis on the IDL definition from the DLMM Swap program indicates the placement of two accounts of event_authority and memo_program should be reversed. Note the related IDL definition can be located at the following LBUZKhRxPF3XUpBCjp4YzTKgLccjZhTSDM9YuVaPwxo address.

```
    // Execute direct DLMM swap with actual swap amount (after fee deduction)
    let accounts = dlmm::cpi::accounts::Swap2 {
        lb_pair: ctx.accounts.lb_pair.to_account_info(),
        bin_array_bitmap_extension: ctx
            .accounts
            .bin_array_bitmap_extension
            .as_ref()
            .map(account account.to_account_info()),
        reserve_x: ctx.accounts.reserve_x.to_account_info(),
        reserve_y: ctx.accounts.reserve_y.to_account_info(),
        user_token_in: ctx.accounts.user_token_in.to_account_info(),
        user_token_out: ctx.accounts.user_token_out.to_account_info(),
        token_x_mint: ctx.accounts.token_x_mint.to_account_info(),
        token_y_mint: ctx.accounts.token_y_mint.to_account_info(),
        oracle: ctx.accounts.oracle.to_account_info(),
        host_fee_in: ctx
            .accounts
            .host_fee_in
            .as_ref()
            .map(account account.to_account_info()),
        user: ctx.accounts.user.to_account_info(),
        token_x_program: ctx.accounts.token_x_program.to_account_info(),
        token_y_program: ctx.accounts.token_y_program.to_account_info(),
        event_authority: ctx.accounts.event_authority.to_account_info(),
        memo_program: ctx.accounts.memo_program.to_account_info(),
        program: ctx.accounts.dlmm_program.to_account_info(),
```

```
    };
```

Listing 3.6: `DlmmSwap::handle_dlmm_swap()`

Recommendation Revise the above-mentioned method to ensure the consistency between the prepared instruction's `accounts` and the defined `IDL` interface.

Status The issue has been fixed in the following commit: a591910.

3.7 Improper Refunded Flag Update Upon Successful User Claims

- ID: PVE-007
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `ClaimUserRewards`
- Category: Business Logic [6]
- CWE subcategory: CWE-837 [3]

Description

Upon successful migration, participating users may claim their tokens and excess, if any. If the fund raise is not successful, participating users may request to refund. While reviewing the claim logic, we notice it improperly sets the refund flag to true.

```
        LaunchStatus::Migrated => {
            // For successful/migrated pools, distribute tokens and excess SOL
            let tokens_to_claim = calculate_user_token_allocation(
                user_position.contributed_sol,
                pool.raised_sol,
                pool.sale_allocation,
            )?;

            // Calculate excess SOL to claim
            let excess_sol_to_claim = if pool.excess_sol > 0 && !user_position.
                excess_sol_claimed {
                user_position.calculate_excess_sol(pool.excess_sol, pool.raised_sol)?
            } else {
                0
            };

            msg!("User claiming: {} tokens, {} excess SOL", tokens_to_claim,
                excess_sol_to_claim);

            // Transfer tokens to user
            if tokens_to_claim > 0 {
                user_position.refunded = true;
```

```
                token::transfer(
                    CpiContext::new_with_signer(
                        ctx.accounts.token_program.to_account_info(),
                        Transfer {
                            from: ctx.accounts.pool_token_vault.to_account_info(),
                            to: ctx.accounts.user_token_account.to_account_info(),
                            authority: ctx.accounts.vault_authority.to_account_info(),
                        },
                        signer_seeds,
                    ),
                    tokens_to_claim,
                )?;
            }

            // Transfer excess SOL to user
            if excess_sol_to_claim > 0 {
                token::transfer(
                    CpiContext::new_with_signer(
                        ctx.accounts.token_program.to_account_info(),
                        Transfer {
                            from: ctx.accounts.pool_quote_vault.to_account_info(),
                            to: ctx.accounts.user_quote_account.to_account_info(),
                            authority: ctx.accounts.vault_authority.to_account_info(),
                        },
                        signer_seeds,
                    ),
                    excess_sol_to_claim,
                )?;
            }

            // Update user position
            user_position.tokens_claimed = true;
            if excess_sol_to_claim > 0 {
                user_position.excess_sol_claimed = true;
            }
            user_position.last_updated = current_time;

            // Emit rewards claimed event
            emit!(UserRewardsClaimed {
                pool: pool.key(),
                user: ctx.accounts.user.key(),
                token_mint: pool.token_mint,
                tokens_claimed: tokens_to_claim,
                excess_sol_claimed: excess_sol_to_claim,
                user_contribution: user_position.contributed_sol,
                pool_total_raised: pool.raised_sol,
                timestamp: current_time,
            });

            msg!("User rewards claimed successfully");
        },
```

Listing 3.7: ClaimUserRewards::claim_user_rewards()

To elaborate, we show above the code snippet of the related branch when handling the successful migration branch inside the `claim_user_rewards()` routine. While it has a rather straightforward logic in claiming the tokens (as well as excess if any), it does improperly set `user_position.refunded = true` (line 153), which should be removed.

Recommendation Revisit the above routine to properly update the refund flag when the migration is successful.

Status The issue has been fixed in the following commit: a591910.

3.8 Trust Issue of Admin Keys

- ID: PVE-008
- Severity: Medium
- Likelihood: Low
- Impact: High
- Target: `Multiple Contracts`
- Category: Security Features [4]
- CWE subcategory: CWE-287 [2]

Description

In the audited protocol, there is a privileged account, i.e., `admin`. This account plays a critical role in governing and regulating the protocol-wide operations (e.g., configure parameters, collect position fees, and execute privileged operations). Our analysis shows that this privileged account needs to be scrutinized. In the following, we examine the privileged account and the related privileged accesses in current contracts.

```
pub fn update_config(
    ctx: Context<UpdateConfig>,
    params: UpdateConfigParams,
) -> Result<()> {
    let config = &mut ctx.accounts.global_config;

    // Update configuration parameters
    if let Some(points_signer) = params.points_signer {
        config.points_signer = points_signer;
    }

    if let Some(points_per_sol) = params.points_per_sol {
        config.points_per_sol = points_per_sol;
    }

    if let Some(min_target_sol) = params.min_target_sol {
        config.min_target_sol = min_target_sol;
    }

```

```
    if let Some(max_target_sol) = params.max_target_sol {
        config.max_target_sol = max_target_sol;
    }

    if let Some(min_duration) = params.min_duration {
        config.min_duration = min_duration;
    }

    if let Some(max_duration) = params.max_duration {
        config.max_duration = max_duration;
    }

    if let Some(paused) = params.paused {
        config.paused = paused;
    }

    if let Some(min_stake_duration) = params.min_stake_duration {
        config.min_stake_duration = min_stake_duration;
    }

    if let Some(lb_pair) = params.lb_pair {
        config.lb_pair = lb_pair;
    }

    msg!("Global config updated successfully");

    Ok(())
}
```

Listing 3.8: Privileged Functions in `update_config()`

We understand the need of the privileged functions for proper operations, but at the same time the extra power to the `admin` may also be a counter-party risk to the protocol users. Therefore, we list this concern as an issue here from the audit perspective and highly recommend making these privileges explicit or raising necessary awareness among protocol users.

Recommendation Promptly transfer the privileged account to the intended `DAO`-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been mitigated as the team plans to assign admin role to a multi-sig wallet.

4 Conclusion

In this audit, we have analyzed the design and implementation of the `Orama Launchpad` protocol, which is committed to building next-generation tokenized asset protocol, aiming to solve key pain points in traditional scientific research—inefficient funding allocation and resource distribution. The platform establishes a closed-loop ecosystem from research to commercialization by funding scientific experiments, enabling intellectual property rights verification, resolving data silos, and implementing community governance, thereby driving innovation in on-chain research. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

[2] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[3] MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. https://cwe.mitre.org/data/definitions/837.html.

[4] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[5] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[6] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[7] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[8] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[9] PeckShield. PeckShield Inc. https://www.peckshield.com.