Pump Science Security Review

Pashov Audit Group

Conducted by: defsec, Koolex, zhaojio

December 24th 2024 - January 2nd 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **moleculeprotocol/pump-science-contract** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Pump Science

Pump Science is a Solana-based framework that facilitates token launches with a bonding curve, dynamic fee structures and automated liquidity management. It works by using a constant product bonding curve where token and SOL reserves dynamically adjust based on purchases.

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 54daf1b93cf6abf955c69f043f73b4df671f97f7

fixes review commit hash - e730b6b1a4852d35d05ca3019e42e29b93c9c3d1

Scope

The following smart contracts were in scope of the audit:

- `util.rs`
- `lib.rs`
- `events.rs`
- `errors.rs`
- `constants.rs`
- `whitelist.rs`
- `mod.rs`
- `meteora.rs`
- `global.rs`
- `structs.rs`
- `mod.rs`
- `locker.rs`
- `curve.rs`
- `create_pool.rs`
- `lock_pool.rs`
- `create_bonding_curve.rs`
- `mod.rs`
- `swap.rs`
- `add_wl.rs`
- `initialize.rs`
- `remove_wl.rs`
- `set_params.rs`

7. Executive Summary

Over the course of the security review, defsec, Koolex, zhaojio engaged with Molecule AG to review Pump Science. In this period of time a total of **11** issues were uncovered.

Protocol Summary

Protocol Name	Pump Science
Repository	https://github.com/moleculeprotocol/pump-science-contract
Date	December 24th 2024 - January 2nd 2024
Protocol Type	Bonding Curve fundraising

Findings Count

Severity	**Amount**
Critical	1
High	3
Medium	2
Low	5
Total Findings	**11**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Lock pool can be DoSed	Critical	Resolved
[H-01]	Direct SOL transfers to bonding curve escrow can break protocol invariant	High	Resolved
[H-02]	Bonding Curve DOS through escrow pre-funding	High	Resolved
[H-03]	Unused freeze authority revocation creates token lock risk	High	Resolved
[M-01]	Start slot validation allows past slots	Medium	Resolved
[M-02]	DOS of CreateBondingCurve	Medium	Resolved
[L-01]	Insufficient SOL balance check ignores rent exemption	Low	Resolved
[L-02]	Insufficient validation in global parameter updates	Low	Resolved
[L-03]	State Inconsistency Due to Solana Rollback	Low	Resolved
[L-04]	Insufficient Token Account validation in pool initialization	Low	Resolved
[L-05]	The Global Account may be initialized early by the attacker	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Lock pool can be DoSed

Severity

Impact: High

Likelihood: High

Description

In the lock_pool instruction, the creation of an Associated Token Account (ATA) for LP tokens relies on checking the lamport balance to determine if the account exists. The check `ctx.accounts.escrow_vault.get_lamports() == 0` is used to decide whether to create the ATA.

```
if ctx.accounts.escrow_vault.get_lamports() == 0 {
    associated_token::create(CpiContext::new(
        ctx.accounts.associated_token_program.to_account_info(),
        associated_token::Create {
            payer: ctx.accounts.payer.to_account_info(),
            associated_token: ctx.accounts.escrow_vault.to_account_info(),
            authority: ctx.accounts.lock_escrow.to_account_info(),
            mint: ctx.accounts.lp_mint.to_account_info(),
            token_program: ctx.accounts.token_program.to_account_info(),
            system_program: ctx.accounts.system_program.to_account_info(),
        },
    ))?;
}
```

programs/pump-science/src/instructions/migration/lock_pool.rs#L157

This is problematic because:

- An attacker can prevent ATA creation by sending SOL to the escrow_vault address beforehand
- Having SOL in an address doesn't guarantee a properly initialized Token Account
- Without a properly initialized ATA, the locking mechanism would fail since there would be no valid token account to receive the LP tokens

As a result, the lock pool would be DoSed.

Recommendations

Replace the current implementation with `create_idempotent` which safely handles ATA creation regardless of lamport balance.

8.2. High Findings

[H-01] Direct SOL transfers to bonding curve escrow can break protocol invariant

Severity

Impact: High

Likelihood: Medium

Description

The bonding curve protocol maintains an invariant that the `real_sol_reserves` state variable must exactly match the SOL balance (lamports) in the `bonding_curve_sol_escrow` account. This invariant is checked at the end of every swap operation:

```
if sol_escrow_lamports != bonding_curve.real_sol_reserves {
    msg!(
        "real_sol_r:{}, bonding_lamps:{}",
        bonding_curve.real_sol_reserves,
        sol_escrow_lamports
    );
    msg!("Invariant failed: real_sol_reserves != bonding_curve_pool_lamports");
    return Err(ContractError::BondingCurveInvariant.into());
}
```

However, the `real_sol_reserves` is only updated during swap operations, while the escrow account's SOL balance can be modified externally through direct transfers. An external SOL transfer to the escrow would increase `sol_escrow_lamports` without updating `real_sol_reserves`, causing the invariant check to fail and making the protocol unusable until fixed.

Note: check the other issue "DOS Attack Vector on Bonding Curve Creation Through Escrow Pre-funding" as it is similiar.

Recommendations

Add a cleanup function that can synchronize `real_sol_reserves` with the actual lamport balance if needed.

[H-02] Bonding Curve DOS through escrow pre-funding

Severity

Impact: High

Likelihood: Medium

Description

The sol_escrow account can be preemptively funded with SOL by an attacker before the bonding curve is created. Since the `create_bonding_curve` instruction initializes `real_sol_reserves` to 0, but the invariant check verifies that the actual SOL balance matches this value, the presence of any SOL in the escrow account will cause the curve creation to fail.

This is possible because:

- The sol_escrow PDA address is deterministic and can be calculated by anyone who knows the mint address
- Anyone can send SOL to this address before the curve is created
- The invariant strictly enforces `sol_escrow_lamports == real_sol_reserves`

Code snippets:

```
// In create_bonding_curve.rs
pub fn handler(
  ctx:Context<CreateBondingCurve>,
  params:CreateBondingCurveParams
) -> Result<(
    // real_sol_reserves initialized to 0
    ctx.accounts.bonding_curve.update_from_params(...);

    // Invariant check will fail if escrow has SOL
    BondingCurve::invariant(locker)?;
}

// In curve.rs
pub fn invariant<'info>(ctx: &mut BondingCurveLockerCtx<'info>) -> Result<()> {
    if sol_escrow_lamports != bonding_curve.real_sol_reserves {
        return Err(ContractError::BondingCurveInvariant.into());
    }
}
```

Recommendations

Initialize `real_sol_reserves` to match any existing SOL balance in the escrow during creation.

If that's not desirable, add a SOL sweep mechanism during curve creation that transfers any existing SOL to the creator, for example:

- Check if there's any existing SOL in the escrow
- If found, sweep it to the creator/admin using a signed CPI
- Then continue with regular curve creation

[H-03] Unused freeze authority revocation creates token lock risk

Severity

Impact: High

Likelihood: Medium

Description

The codebase includes a `revoke_freeze_authority` function in `locker.rs` that is not being called during pool creation, despite being necessary to prevent tokens from being frozen after migration.

If the freeze authority is not revoked:

- The bonding curve program retains the ability to freeze token accounts
- This could potentially be used to lock user tokens after migration
- Goes against the intended design where locking should only be possible during pre-trading

Code Location : locker.rs#L83

Recommendations

It's recommended that the freeze authority revocation call be added.

8.3. Medium Findings

[M-01] Start slot validation allows past slots

Severity

Impact: Medium

Likelihood: Medium

Description

The protocol implements a fee schedule that should progress from high fees (99%) to low fees (1%) over time based on slots passed since launch. However, the current implementation validates the `start_slot` parameter incorrectly:

```
pub fn validate(&self, params: &CreateBondingCurveParams) -> Result<()> {
    let clock = Clock::get()?;
    // validate start time
    if let Some(start_slot) = params.start_slot {
        require!(start_slot <= clock.slot, ContractError::InvalidStartTime)
    }
    // ...
}
```

This validation only ensures the start_slot is not in the future (`start_slot <= clock.slot`), meaning creators can set a start slot far in the past. This breaks the intended fee progression since `slots_passed = current_slot - start_slot` would immediately start at a later fee stage.

For example: The current slot is 1000

- Creator sets start_slot to 800 The first trade occurs at slot 1000, slots_passed = 1000 - 800 = 200
- This means trading starts in Phase 2 (~5% fee) instead of Phase 1 (99% fee)

Recommendations

The start slot validation should be updated to only allow start slots within a small range of the current slot, ensuring the fee schedule starts from Phase 1 as intended.

[M-02] DOS of CreateBondingCurve

Severity

Impact: Medium

Likelihood: Medium

Description

The `CreateBondingCurve` instruction requires passing `bonding_curve_token_account`:

```
#[account(
        init,
        payer = creator,
        associated_token::mint = mint,
        associated_token::authority = bonding_curve,
    )]
    bonding_curve_token_account: Box<Account<'info, TokenAccount>>,
```

`bonding_curve_token_account` uses `associated_token::mint` and `associated_token::authority`, so the `TokenAccount` can be created in advance. If this already exists, creating the function will fail because init is used.

An attacker can extract the calculated `bonding_curve` address and create a `TokenAccount` to prevent users from creating a Bonding Curve.

Recommendations

````
#[account(
-        init,
+        init_if_needed,
        payer = creator,
        associated_token::mint = mint,
        associated_token::authority = bonding_curve,
    )]
    bonding_curve_token_account: Box<Account<'info, TokenAccount>>,```
````

8.4. Low Findings

[L-01] Insufficient SOL balance check ignores rent exemption

In the swap function's buy tokens validation, the code only checks if the user's total SOL balance is sufficient for the swap amount, without considering rent exemption:

```
require!(
    ctx.accounts.user.get_lamports() >= exact_in_amount,
    ContractError::InsufficientUserSOL,
);
```

programs/pump-science/src/instructions/curve/swap.rs#L177

This check uses `get_lamports()` which returns the total balance including the rent exemption amount. The transaction will succeed even if it depletes the account below rent exemption, causing the account to:

- Enter a non-rent-exempt state
- Get charged rent over time by the Solana runtime, eventually be purged if the balance is fully depleted.

Add a more comprehensive balance check that accounts for rent exemption:

```
let rent = Rent::get()?;
let min_rent = rent.minimum_balance
//(0); // 0 for data size since this is just a native SOL account
require!(
    ctx.accounts.user.get_lamports() >= exact_in_amount.checked_add
      (min_rent).unwrap(),
    ContractError::InsufficientUserSOL,
);
```

[L-02] Insufficient validation in global parameter updates

The `set_params` function allows updating protocol parameters without proper validation:

```
pub fn set_params
  (ctx: Context<SetParams>, params: GlobalSettingsInput) -> Result<()> {
    SetParams::handler(ctx, params)
}
```

Looking at the `GlobalSettingsInput` struct:

```
pub struct GlobalSettingsInput {
    pub initial_virtual_token_reserves: Option<u64>,
    pub initial_virtual_sol_reserves: Option<u64>,
    pub initial_real_token_reserves: Option<u64>,
    pub token_total_supply: Option<u64>,
    pub mint_decimals: Option<u8>,
    pub migrate_fee_amount: Option<u64>,
    pub migration_token_allocation: Option<u64>,
    pub fee_receiver: Option<Pubkey>,
    pub whitelist_enabled: Option<bool>,
    pub meteora_config: Option<Pubkey>,
}
```

Consider adding comprehensive parameter validation.

[L-03] State Inconsistency Due to Solana Rollback

One function in the protocol is vulnerable to state inconsistencies in the event of a Solana rollback:

1. **Setting Config Parameters**:
    - Global configuration parameters could become outdated during a Solana rollback
    - Protocol could operate with old, invalid settings
    - Potential for system malfunction or vulnerabilities

Recommendation:

1. **Detect Outdated Configurations**

    - Utilize the `LastRestartSlot` sysvar to check configuration validity
    - Automatically pause protocol if the configuration is outdated
    - Require admin intervention before resuming operations

2. **Add last_updated_slot Field**

   - Include tracking field in bonding curve state
   - Monitor configuration update timestamps

3. **Implement Outdated Configuration Check**

```
fn is_config_outdated(global: &Global) -> Result<bool> {
    let last_restart_slot = LastRestartSlot::get()?;
    Ok(global.last_updated_slot <= last_restart_slot.last_restart_slot)
}
```

[L-04] Insufficient Token Account validation in pool initialization

Several token accounts in the `InitializePoolWithConfig` instruction lack proper ownership and mint relationship validation, relying on `UncheckedAccount` instead of proper token account validation. This could allow to pass incorrect token accounts.

The current implementation uses `UncheckedAccount` for multiple token-related accounts:

```
#[derive(Accounts)]
pub struct InitializePoolWithConfig<'info> {
    #[account(mut)]
    /// CHECK: Token A LP
    pub a_vault_lp: UncheckedAccount<'info>,

    #[account(mut)]
    /// CHECK: Vault LP accounts and mints for token A
    pub a_token_vault: UncheckedAccount<'info>,

    // ... other unchecked token accounts
}
```

This lacks validation for:

1. Token account ownership verification
2. Mint relationship verification
3. Type safety provided by Anchor's account validation

Add proper token account validation:

```
#[derive(Accounts)]
pub struct InitializePoolWithConfig<'info> {
    #[account(
        mut,
        constraint = a_vault_lp.owner == vault_program.key
          () @ ContractError::InvalidOwner,
        constraint = a_vault_lp.mint == a_vault_lp_mint.key
          () @ ContractError::InvalidMint
    )]
    pub a_vault_lp: Box<Account<'info, TokenAccount>>,

    #[account(
        mut,
        constraint = a_token_vault.owner == vault_program.key
          () @ ContractError::InvalidOwner,
        constraint = a_token_vault.mint == token_a_mint.key
          () @ ContractError::InvalidMint
    )]
    pub a_token_vault: Box<Account<'info, TokenAccount>>,

    // Similar changes for other token accounts
}
```

Add new error variants:

```
#[error_code]
pub enum ContractError {
    #[msg("Invalid token account owner")]
    InvalidOwner,

    #[msg("Invalid token account mint")]
    InvalidMint,
}
```

Apply similar validation to all token accounts:

- Vault LP accounts (a_vault_lp, b_vault_lp)
- Token vault accounts (a_token_vault, b_token_vault)
- Payer token accounts (payer_token_a, payer_token_b)
- Protocol fee accounts (protocol_token_a_fee, protocol_token_b_fee)

[L-05] The Global Account may be initialized early by the attacker

Severity

Impact: Medium

Likelihood: Low

Description

The Global Account uses a fixed string (Global::SEED_PREFIX) as its seeds, and anyone can call the initialization function as long as `global.initialized` is not set.

```
#[event_cpi]
#[derive(Accounts)]
#[instruction(params: GlobalSettingsInput)]
pub struct Initialize<'info> {
    #[account(mut)]
    authority: Signer<'info>,

    #[account(
        init,
        space = 8 + Global::INIT_SPACE,
@>    seeds = [Global::SEED_PREFIX.as_bytes()],
@>
      constraint = global.initialized != true @ ContractError::AlreadyInitialized,
        bump,
        payer = authority,
    )]
    global: Box<Account<'info, Global>>,

    system_program: Program<'info, System>,
}

impl Initialize<'_> {
    pub fn handler
      (ctx: Context<Initialize>, params: GlobalSettingsInput) -> Result<()> {
        let global = &mut ctx.accounts.global;
        global.update_authority(GlobalAuthorityInput {
            global_authority: Some(ctx.accounts.authority.key()),
            migration_authority: Some(ctx.accounts.authority.key()),
        });
        global.update_settings(params.clone());

        require_gt!(global.mint_decimals, 0, ContractError::InvalidArgument);

        global.initialized = true;
        emit_cpi!(global.into_event());
        Ok(())
    }
}
```

Therefore, an attacker can preemptively call the initialization function to set its own address as the administrator. As a result, the Global Account cannot be reset.

Recommendations

Call the signer's address as seeds, or just run the fixed address call initialization function.