Desci Launchpad Security Review

Pashov Audit Group

Conducted by: defsec, FrankCastle, ZanyBonzy

February 7th 2025 - February 10th 2025

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 **merklelabshq/desci-launchpad** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Desci Launchpad

Desci Launchpad is a system for launching and funding projects, involving curation, acceleration, and separation. It enables participants to back projects, with raised funds creating liquidity pools and offering community benefits.

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 - 7167b472dd4e1c1a45ed2d49f00cd1dfadad0fcd

fixes review commit hash - e41b734a8280fb7fe6440f9daf5f647b6ac5dec9

Scope

The following smart contracts were in scope of the audit:

- `buy_token.rs`
- `claim_revenue`
- `claim_token`
- `create_token`
- `deposit_token`
- `init_stats`
- `mod`
- `update_token`
- `withdraw_tokenstate`
- `lib`
- `error`
- `constants`

7. Executive Summary

Over the course of the security review, defsec, FrankCastle, ZanyBonzy engaged with Merkle Labs to review Desci Launchpad. In this period of time a total of **12** issues were uncovered.

Protocol Summary

Protocol Name	Desci Launchpad
Repository	https://github.com/merklelabshq/desci-launchpad
Date	February 7th 2025 - February 10th 2025
Protocol Type	Launchpad

Findings Count

Severity	**Amount**
Critical	1
High	2
Medium	1
Low	8
Total Findings	**12**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Unclaimed tokens locked in stats_token causing permanent fund loss	Critical	Resolved
[H-01]	withdraw_tokens fails to adjust claimed_supply locking tokens permanently	High	Resolved
[H-02]	claim_revenue lets admin block user withdrawals below min threshold	High	Resolved
[M-01]	Prevent token claims if the minimum threshold is not surpassed	Medium	Resolved
[L-01]	Incorrect check for > 0 buy amt	Low	Resolved
[L-02]	Missing maximum cooldown duration validation	Low	Resolved
[L-03]	Unused is_lp_created flag	Low	Resolved
[L-04]	Missing toolchain version in Anchor.toml	Low	Resolved
[L-05]	Use of transfer instead of transfer_checked	Low	Resolved
[L-06]	All administrative keys are identical	Low	Acknowledged
[L-07]	State inconsistency due to solana rollback	Low	Acknowledged
[L-08]	Missing validation for start_time and end_time	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Unclaimed tokens locked in `stats_token` causing permanent fund loss

Severity

Impact: High

Likelihood: High

Description

In the `claim_token` function, tokens are distributed to users based on the proportion of tokens they purchased relative to the total `claimed_supply`. If the `claimed_supply` is less than the `sale_supply`, users receive a proportional amount of `claimed_supply` rather than the full `sale_supply`.

This is reflected in the following code snippet:

```
let adjusted_tokens = (
  (user_stats.tokens_purchased as f64 / token_stats.claimed_supply as f64)
    * (token_stats.sale_supply.min(token_stats.claimed_supply) as f64))
    as u64;

transfer(
    CpiContext::new_with_signer(
        ctx.accounts.token_program.to_account_info(),
        Transfer {
            from: ctx.accounts.stats_token.to_account_info(),
            to: ctx.accounts.user_token.to_account_info(),
            authority: ctx.accounts.stats.to_account_info(),
        },
        &[signer_seed],
    ),
    adjusted_tokens,
)?;
```

Since the amount distributed from `stats_token` is less than the `sale_supply` originally transferred to the `stats` account in `deposit_token`, as shown below:

```
pub fn deposit_token_handler(ctx: Context<DepositToken>) -> Result<()> {
    let token_stats = &mut ctx.accounts.token_stats;
    transfer(
        CpiContext::new(
            ctx.accounts.token_program.to_account_info(),
            Transfer {
                from: ctx.accounts.authority_token.to_account_info(),
                to: ctx.accounts.stats_token.to_account_info(),
                authority: ctx.accounts.authority.to_account_info(),
            },
        ),
        token_stats.sale_supply,
    )?;
```

This results in the difference between `sale_supply` and `claimed_supply` being locked indefinitely in the `stats_token` account, causing a permanent loss of funds for the protocol.

Recommendations

To mitigate this issue, consider implementing one of the following solutions:

1. Introduce a function to transfer the difference between `sale_supply` and `claimed_supply` back to the authority account if any surplus exists.
2. Modify the `claim_revenue` function to account for this difference and transfer the remaining tokens accordingly.

8.2. High Findings

[H-01] `withdraw_tokens` fails to adjust `claimed_supply` locking tokens permanently

Severity

Impact: High

Likelihood: Medium

Description

The `withdraw_tokens` function allows users to reclaim their payment tokens if the `min_threshold` is not met. However, the function does not update `claimed_supply` or transfer `tokens_purchased` back to the authority to reflect the refunded amounts.

Since the user's stats are marked as claimed, the corresponding `tokens_purchased` remain locked indefinitely.

For example, if a user has `tokens_purchased = 1000` and the `claimed_supply = 5000`, after withdrawal, and then he call the `withdraw_tokens` function and get his payment tokens back and only `4000` tokens can be claimed by others, while `1000` tokens remain permanently locked.

The relevant code snippet is shown below:

```rust
pub fn withdraw_token_handler(ctx: Context<WithdrawToken>) -> Result<()> {
    let token_stats = &mut ctx.accounts.token_stats;
    let user_stats: &mut Account<UserStats> = &mut ctx.accounts.user_stats;

    let curr_time = Clock::get()?.unix_timestamp;
    require!(
        token_stats.end_time < curr_time,
        RocketxLaunchpadError::InvalidWithdrawTime
    );
    require!(
        token_stats.is_launched,
        RocketxLaunchpadError::TokenNotLaunched
    );
    require!(
        !user_stats.is_claimed,
        RocketxLaunchpadError::TokenAlreadyClaimed
    );
    require!(
        token_stats.revenue < token_stats.min_threshold,
        RocketxLaunchpadError::InvalidThreshold
    );

    user_stats.is_claimed = true;

    let signer_seed = &[STATS_SEED, &[ctx.accounts.stats.bump]];

    let refund_amount = (((user_stats.tokens_purchased as f64)
        / (10u64.pow(token_stats.decimals as u32) as f64))
        * (token_stats.price_per_token as f64)) as u64;

    transfer(
        CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            Transfer {
                from: ctx.accounts.stats_pay_token.to_account_info(),
                to: ctx.accounts.user_pay_token.to_account_info(),
                authority: ctx.accounts.stats.to_account_info(),
            },
            &[signer_seed],
        ),
        refund_amount,
    )?;

    Ok(())
}
```

Recommendations

To properly reflect the token refund and prevent permanent token locking, consider the following mitigations:

1. **Subtract** `tokens_purchased` **from** `claimed_supply` when a user withdraws their payment tokens.
2. **Transfer the corresponding** `tokens_purchased` **amount back to the authority** to ensure the tokens are not locked indefinitely.

[H-02] `claim_revenue` lets admin block user withdrawals below min threshold

Severity

Impact: High

Likelihood: Medium

Description

The `claim_revenue` function should not be executable if the revenue falls below the `min_threshold`. In this case, users are allowed to reclaim their payment tokens after the sale duration ends. However, if the admin calls the `claim_revenue` function immediately after the sale ends, users will no longer be able to withdraw their payment tokens via `withdraw_token`.

The `withdraw_token` function requires the revenue to be less than `min_threshold` to enable withdrawals, as shown in the code snippet below:

```
pub fn withdraw_token_handler(ctx: Context<WithdrawToken>) -> Result<()> {
    require!(
        token_stats.is_launched,
        RocketxLaunchpadError::TokenNotLaunched
    );

    require!(
        !user_stats.is_claimed,
        RocketxLaunchpadError::TokenAlreadyClaimed
    );

    require!(
        token_stats.revenue < token_stats.min_threshold,
        RocketxLaunchpadError::InvalidThreshold
    );
```

If the admin withdraws the revenue, users will be unable to withdraw their payment tokens. Additionally, if a user withdraws some payment tokens, the admin will not be able to call `claim_revenue` because the balance of `stats_pay_token` will be lower than `token_stats.revenue`, causing the transfer to fail.

Recommendations

Consider implementing one of the following fixes:

1. **Disable `claim_revenue` if the launchpad has not reached the minimum revenue threshold**, allowing only users to reclaim their payment tokens.
2. **Introduce a cooldown period for `claim_revenue`**, giving users time to withdraw their payment tokens before the admin can claim revenue. Modify `claim_revenue` to transfer only the remaining payment token balance in the `stats_pay_token` account.

8.3. Medium Findings

[M-01] Prevent token claims if the minimum threshold is not surpassed

Severity

Impact: Medium

Likelihood: Medium

Description

Allowing users to either **claim tokens** or **withdraw payment tokens** when the `min_threshold` is not met can result in locked funds. If some users claim their tokens while others withdraw, this creates a scenario where uncollected payments remain in the contract, leading to fund loss.

To prevent this, token claims should be restricted if the minimum revenue threshold has not been surpassed.

Recommendations

Add the following check to the `claim_tokens` function to ensure tokens can only be claimed when the revenue meets or exceeds the minimum threshold:

```
require!(
    token_stats.revenue >= token_stats.min_threshold,
    RocketxLaunchpadError::InvalidThreshold
);
```

8.4. Low Findings

[L-01] Incorrect check for > 0 buy amt

Describe the finding and your recommendation here

`buy_token` incorrectly checks that `token_amount` is greater than 0, rather than checking that `buy_amount` is > 0. As a result, 0 amount of tokens can be bought.

```
require!(args.token_amount > 0, RocketxLaunchpadError::InvalidAmount);
```

Recommend refactoring the check to check for `buy_amount` instead. This also fixes the check below since `buy_amount` is first set to the min of `token_amount` and `limit_per_wallet - tokens_purchased`

```
require!(
        user_stats.tokens_purchased <= token_stats.limit_per_wallet,
        RocketxLaunchpadError::ExceedsLimit
    );
```

[L-02] Missing maximum cooldown duration validation

The `cooldown_duration` value provided in `args.cooldown_duration` is directly assigned to `token_stats.cooldown_duration` without any validation. This allows users to set an excessively long cooldown period, which could negatively impact the protocol's usability and fairness.

```
token_stats.cooldown_duration = args.cooldown_duration;
```

Introduce a **maximum cooldown period** by adding a check to ensure that `cooldown_duration` does not exceed a predefined upper limit.

Example fix:

```
require!(
    args.cooldown_duration <= MAX_COOLDOWN_DURATION,
    RocketxLaunchpadError::InvalidCooldownDuration
);

token_stats.cooldown_duration = args.cooldown_duration;
```

[L-03] Unused `is_lp_created` flag

The `token_stats.is_lp_created` flag is updated in the `update_token_handler` function but is not used elsewhere in the contract. This could indicate unnecessary state storage, leading to redundant data or potential confusion in the contract logic.

```
pub fn update_token_handler(ctx: Context<UpdateToken>) -> Result<()> {
    let token_stats = &mut ctx.accounts.token_stats;

    token_stats.is_lp_created = true;

    Ok(())
}
```

Consider either utilizing the `is_lp_created` flag in relevant logic or removing it if it is not required.

[L-04] Missing toolchain version in Anchor.toml

The `Anchor.toml` file does not specify the `anchor_version` or `solana_version` under the `[toolchain]` section. This can lead to compatibility issues when building or deploying the program, especially if different team members or CI/CD pipelines use different versions of Anchor or Solana.

```
[toolchain]

[features]
resolution = true
skip-lint = false

[programs.devnet]
desci_launchpad = "BtUNgrRufngrdEbXMnqUQjWP5LhKGGkz89U75k5tuaSj"

[registry]
url = "https://api.apr.dev"

[provider]
cluster = "Devnet"
wallet = "~/.config/solana/id.json"

[scripts]
deploy
   = "anchor build && solana program deploy ./target/deploy/desci_launchpad.so --skip-
test
   = "yarn run ts-mocha -p ./tsconfig.json -t 1000000 tests/desci-launchpad.ts"
```

Recommendation:

Add the `anchor_version` and `solana_version` to the `[toolchain]` section to ensure consistent builds and deployments.

[L-05] Use of `transfer` instead of `transfer_checked`

In the `buy_token_handler` function and other function, the `transfer` instruction is used to transfer tokens from the user's payment token account to the stats payment token account. However, `transfer` does not validate the mint account or the number of decimals, which can lead to potential issues:

1. **Mint Mismatch**: If the wrong mint account is passed, the transfer could succeed but involve the wrong token.
2. **Decimal Mismatch**: If the number of decimals is incorrect, the amount transferred could be different from what was intended (e.g., sending 1,000,000 tokens instead of 10 due to a decimal mismatch).

Replace the `transfer` instruction with `transfer_checked`, which requires the mint account and the number of decimals to be passed.

[L-06] All administrative keys are identical

The constants `DEV_PUBKEY`, `ADMIN_PUBKEY`, and `MINT_AUTHORITY_PUBKEY` are all set to the same value:

```
pub const DEV_PUBKEY: Pubkey = pubkey!
  ("BjjFpCbTrFVn3ZgcdCv4jTLAzbbDCMV1Vo115XJSJ7XG");
pub const ADMIN_PUBKEY: Pubkey = pubkey!
  ("BjjFpCbTrFVn3ZgcdCv4jTLAzbbDCMV1Vo115XJSJ7XG");
pub const MINT_AUTHORITY_PUBKEY: Pubkey = pubkey!
  ("BjjFpCbTrFVn3ZgcdCv4jTLAzbbDCMV1Vo115XJSJ7XG");
```

This design poses a risk because it centralizes all administrative privileges into a single key. If this key is compromised, an attacker could gain control over all administrative functions, including:

- Development operations (`DEV_PUBKEY`)
- Administrative actions (`ADMIN_PUBKEY`)
- Minting authority (`MINT_AUTHORITY_PUBKEY`)

If the shared key is compromised, the entire system is at risk. An attacker could perform unauthorized actions, such as minting tokens, modifying configurations, or accessing sensitive data.

Recommendation:

Assign unique public keys to `DEV_PUBKEY`, `ADMIN_PUBKEY`, and `MINT_AUTHORITY_PUBKEY` to ensure role separation.

[L-07] 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 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-08] Missing validation for `start_time` and `end_time`

The `create_token_handler` function does not validate whether the `start_time` and `end_time` provided in the `CreateTokenArgs` are in the future relative to the current time.

This can lead to the following issue:

Invalid Launchpad Timing: If `start_time` or `end_time` is in the past, the token sale may start or end immediately.

Code Location : create_token.rs

```
token_stats.token_id = ctx.accounts.stats.tokens_created;
    token_stats.name = args.name.clone();
    token_stats.symbol = args.symbol.clone();
    token_stats.uri = args.uri.clone();
    token_stats.decimals = args.decimals;
    token_stats.payment_token = args.payment_token.key();
    token_stats.total_supply = args.total_supply;
    token_stats.sale_supply = args.sale_supply;
    token_stats.limit_per_wallet = args.limit_per_wallet;
    token_stats.price_per_token = args.price_per_token;
    token_stats.start_time = args.start_time;
    token_stats.end_time = args.end_time;
    token_stats.cooldown_duration = args.cooldown_duration;
    token_stats.min_threshold = args.min_threshold;
    token_stats.max_threshold = args.max_threshold;
    token_stats.bump = ctx.bumps.token_stats;
```

Add validation to ensure that both `start_time` and `end_time` are in the future relative to the current time. Use the `Clock::get()?` function to retrieve the current timestamp and compare it with `start_time` and `end_time`.