CD SECURITY

RUST AUDIT REPORT

Tajir Media

August 2024

Prepared by
tsvetanovv

Introduction

A time-boxed security review of the **TJRM** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

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.

About **TJRM**

This ICO program on the Solana blockchain facilitates the private and pre-sale phases of a token offering. It allows users to purchase tokens using either SOL or USDT, providing flexibility and accessibility.

The protocol securely handles transactions, managing token distribution and account balances through various functions. These essential functions are:

- Token Management: `create_ico_ata()` initializes accounts and transfers ICO tokens from the admin to the program's ATA (Associated Token Account).
- Purchasing: `buy_with_sol()` and `buy_with_usdt()` allow users to buy ICO tokens using SOL or USDT, respectively, with token calculations based on current prices.
- Administration: Functions like `deposit_ico_in_ata()`, `update_data()`, and `withdraw()` enable the admin or manager to manage the ICO, such as depositing more tokens, updating ICO parameters, or withdrawing remaining tokens after the ICO ends.

Severity classification

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

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 5b53268320410df4462211dec07411ff874fac5c**

Scope

The following smart contracts were in scope of the audit:

- `solana-ico/src/lib.rs`

The following number of issues were found, categorized by their severity:

- Critical & High: 3 issues
- Medium: 2 issues
- Low: 2 issues

Findings Summary

ID	Title	Severity	Status
[C-01]	`deposit_ico_in_ata()` overwrites total supply	Critical	Resolved
[H-01]	Lack of Slippage Protection in `buy_with_sol()` function	High	Resolved
[H-02]	Misuse of SOL/USDC Price Feed for USDT Calculations	High	Resolved
[M-01]	`load_price_feed_from_account_info` is deprecated	Medium	Resolved
[M-02]	Vulnerable Packages	Medium	Resolved
[L-01]	Avoid Hardcoded values	Low	Resolved
[L-02]	Manager can use `withdraw` function during the ICO	Low	Resolved

Detailed Findings

[C-01] `deposit_ico_in_ata()` overwrites the total supply instead of adding to the total amount

Description

The `deposit_ico_in_ata` function allows the admin to add more ICO tokens to the program's associated token account (ATA). This ensures that the ICO can continue with an increased supply of tokens.

```
    pub fn deposit_ico_in_ata(ctx: Context<DepositIcoInATA>, ico_amount:
u64) -> ProgramResult {
        if ctx.accounts.data.admin != *ctx.accounts.admin.key {
            return Err(ProgramError::IncorrectProgramId);
        }
        // transfer ICO admin to program ata
        let cpi_ctx = CpiContext::new(
            ctx.accounts.token_program.to_account_info(),
            token::Transfer {
                from: ctx.accounts.ico_ata_for_admin.to_account_info(),
                to:
ctx.accounts.ico_ata_for_ico_program.to_account_info(),
                authority: ctx.accounts.admin.to_account_info(),
            },
        );
        token::transfer(cpi_ctx, ico_amount)?;
        let data = &mut ctx.accounts.data;
        data.total_amount = ico_amount;
        msg!("deposit {} ICO in program ATA.", ico_amount);
        Ok(())
    }
```

The problem here is that the line `data.total_amount = ico_amount;` in the `deposit_ico_in_ata()` function overrides the `total_amount` field with the new `ico_amount` rather than adding to the existing amount.

This design flaw causes the total token amount to reset with each new deposit, leading to the loss of previous deposit records.

Recommendations

Replace `data.total_amount = ico_amount;` with `data.total_amount += ico_amount;` to correctly accumulate the total ICO token supply across multiple deposits. This ensures accurate tracking of the total tokens available for the ICO.

[H-01] Lack of Slippage Protection in `buy_with_sol()` function

Description

The `buy_with_sol()` function allows users to purchase ICO tokens using SOL.

The function retrieves the current SOL/USDC price from a price feed and calculates the value of the SOL the user is spending in USDT. After this, the number of ICO tokens is determined based on the equivalent USDT value.

The problem here is that this function does not incorporate slippage protection. Given the inherent volatility of cryptocurrency prices, this absence could result in users paying more or receiving fewer tokens than expected if the price fluctuates between the time the price is fetched and the transaction is executed.

Recommendations

Introduce a maximum allowable slippage percentage that users can set before executing the transaction.

[H-02] Misuse of SOL/USDC Price Feed for USDT Calculations

Description

The code uses the `SOL_USDC_FEED` to fetch the price of SOL in USDC:

```
const SOL_USDC_FEED: &str =
"J83w4HKfqxwcq3BEMMkPFSppX3gqekLyLJBexebFVkix";
```

And uses this price to perform calculations in USDT:

```
let current_price = price_feed.get_price_no_older_than(current_timestamp,
STALENESS_THRESHOLD);
```

While USDC and USDT are both stablecoins, they are distinct assets and do not always maintain a 1:1 value, especially during market volatility. A historical example is the USDC depeg in 2023 when it dropped to 0.80 USD.

If such a depeg occurs again, it could be exploited by malicious users to purchase ICO tokens at a significantly reduced rate, leading to financial losses for the project.

Recommendations

Replace the SOL/USDC feed with a direct SOL/USDT price feed to ensure accurate and relevant price calculations.

[M-01] `load_price_feed_from_account_info` is deprecated

Description

The function `load_price_feed_from_account_info` in the Pyth SDK for Solana has been deprecated.

```
use pyth_sdk_solana::{load_price_feed_from_account_info, PriceFeed};
```

The function has been deprecated since version `0.10.x` of `pyth_sdk_solana`.

In dependencies, we see that the protocol currently uses: `pyth-sdk-solana = "0.10.1"`. This means that this function is no longer recommended for use.

Reference - Link

Recommendations

The recommended alternative is to use `SolanaPriceAccount::account_info_to_feed` instead.

[M-02] Vulnerable Packages

Description

Currently, the protocol uses two vulnerable packages.

Timing variability in `curve25519-dalek`'s `Scalar29::sub`/`Scalar52::sub`

Timing variability of any kind is problematic when working with potentially secret values such as elliptic curve scalars, and such issues can potentially leak private keys and other secrets.

Reference - https://rustsec.org/advisories/RUSTSEC-2024-0344

Double Public Key Signing Function Oracle Attack on `ed25519-dalek

Versions of `ed25519-dalek` prior to v2.0 model private and public keys as separate types which can be assembled into a `Keypair`, and also provide APIs for serializing and deserializing 64-byte private/public keypairs.

Such APIs and serializations are inherently unsafe as the public key is one of the inputs used in the deterministic computation of the `S` part of the signature, but not in the `R` value. An adversary could somehow use the signing function as an oracle that allows arbitrary public keys as input can obtain two signatures for the same message sharing the same `R` and only differ on the `S` part.

Unfortunately, when this happens, one can easily extract the private key.

Reference - https://rustsec.org/advisories/RUSTSEC-2022-0093

Recommendations

Have a look in the reference in which version the vulnerabilities are fixed and update at least to that version.

[L-01] Avoid Hardcoded values

The protocol uses a hardcoded value to set a minimum threshold of 50 USDT when using the `buy_with_sol()` and `buy_with_usdt()` functions.

```
if((sol_in_usdt) / 1000000000) < 50000000 {
....
if usdt_amount < 50000000 {
```

It is not good practice to use hardcoded values because they can lead to confusion and future problems. In addition, under certain economic circumstances, the admin may decide to increase/decrease the value and this cannot be possible.

Recommendations

Add setter functions for admin to set the values.

[L-02] Manager can use `withdraw` function during the ICO

The `withdraw()` function allows the admin or manager of the ICO program to withdraw any remaining tokens after the ICO has ended:

```
     pub fn withdraw(
        ctx: Context<WithDraw>,
        _ico_ata_for_ico_program_bump: u8,
        token_amount: u64,
    ) -> ProgramResult {
        if ctx.accounts.data.admin != ctx.accounts.admin.key() {
            return Err(ProgramError::IllegalOwner);
        }
        if ctx.accounts.data.manager != ctx.accounts.manager.key() {
            return Err(ProgramError::IllegalOwner);
        }
        let ico_mint_address = ctx.accounts.ico_mint.key();
        let seeds = &[ico_mint_address.as_ref(), &
[_ico_ata_for_ico_program_bump]];
        let signer = [&seeds[..]];
        let cpi_ctx = CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            token::Transfer {
                from:
ctx.accounts.ico_ata_for_ico_program.to_account_info(),
                to: ctx.accounts.ico_ata_for_user.to_account_info(),
                authority:
ctx.accounts.ico_ata_for_ico_program.to_account_info(),
            },
            &signer,
        );
```

```
            token::transfer(cpi_ctx, token_amount)?;
            Ok(())
        }
```

However, there are no checks to ensure that this can only happen after the ICO ends. This means that the admin or manager could potentially withdraw tokens at any time, even during the ICO, which poses a risk of privilege abuse.

Recommendations

You could add a check that ensures the ICO has ended before allowing the withdrawal