
# Definitive

## Security Assessment

May 3rd, 2024 — Prepared by OtterSec

Nicola Vella nick0ve@osec.io

Tamta Topuria tamta@osec.io

Robert Chen notdeghost@osec.io

# Table of Contents

# 01 — Executive Summary

## Overview

Definitive engaged OtterSec to assess the `solana-trading` program. This assessment was conducted between May 7th and May 21st, 2024. For more information on our auditing methodology, refer to Appendix B.

## Key Findings

We produced 9 findings throughout this audit engagement.

In particular, we identified a vulnerability regarding a possible scenario where traders may be able to withdraw all the funds from a group (OS-STP-ADV-00). Furthermore, the `RemoveGroupUser` instruction may lock out the owner of a group by allowing the removal of the group owner, with no method to re-dd them with `Owner` privileges (OS-STP-ADV-04). Additionally, the current logic enables admins to withdraw funds from any group and potentially lock themselves out of the protocol (OS-STP-ADV-06 and OS-STP-ADV-05).

We also recommended setting fee limits to prevent admins from charging excessive fees for swap operations (OS-STP-SUG-00).

# 02 — Scope

The source code was delivered to us in a Git repository at https://github.com/DefinitiveCo/solana-trading. This audit was performed against commit 28091d9.

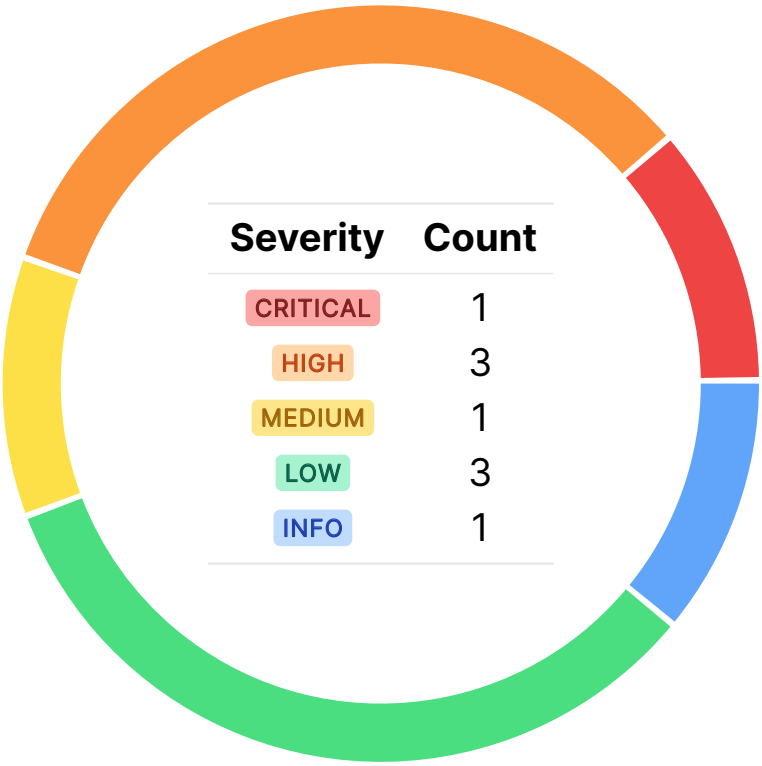
**A brief description of the programs is as follows:**

| Name | Description |
| --- | --- |
| solana-trading | The Solana trading protocol facilitates the management and creation of the trading group and associated user roles and functionalities such as deposits, withdrawals, and swaps. |

# 03 — Findings

Overall, we reported 9 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



| Severity | Count |
| --- | --- |
| CRITICAL | 1 |
| HIGH | 3 |
| MEDIUM | 1 |
| LOW | 3 |
| INFO | 1 |

# 04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

| ID | Severity | Status | Description |
|---|---|---|---|
| OS-STP-ADV-00 | CRITICAL | RESOLVED | A malicious trader may bypass the intended sequence of `PreSwap` and `PostSwap` instructions using a `Cross Program Invocation`, leading to potential withdrawal of all funds. |
| OS-STP-ADV-01 | HIGH | RESOLVED | The `deposit` instruction allows deposits to any token account, but swap instructions require an associated token account for the token. |
| OS-STP-ADV-02 | HIGH | RESOLVED | The `output_group_token_account` and `output_token_account` checks may be bypassed by placing the account in the transaction's `remaining_accounts` list. |
| OS-STP-ADV-03 | HIGH | RESOLVED | An attacker may front-run a legitimate swap by depositing dust to the output token account, resulting in the check failing. |
| OS-STP-ADV-04 | MEDIUM | RESOLVED | `RemoveGroupUser` instruction allows the removal of the group owner without a method of re-adding them with `Owner` privileges via `AddGroupUser`. |
| OS-STP-ADV-05 | LOW | RESOLVED | `update` updates the Definitive admin key without verifying it as a signer. This creates a risk where admins may accidentally lock themselves out. |
| OS-STP-ADV-06 | LOW | RESOLVED | Admins may withdraw funds from any group. |
| OS-STP-ADV-07 | LOW | RESOLVED | Checking for a test swap instruction ID in the production code may result in security issues. |

## Missing CPI Guard CRITICAL OS-STP-ADV-00

### Description

The current implementation of the `swap_introspection_checks` function is vulnerable to a bypass when the program is called through `CPI`. This allows an attacker to circumvent the requirement that `PreSwap` and `PostSwap` must be called together in a single transaction.

### Proof of Concept

By crafting a transaction that includes specific instructions, an attacker can manipulate the program state as follows:

1. Invoke an attacker-controlled program with input data set to the `PreSwap::discriminator`.
2. Within this program, make a CPI to `DefinitiveProgram::PreSwap` with `input_swap_amount` set to the amount they intend to steal and `input_fee_amount` set to zero.
3. Immediately follow with another CPI to `DefinitiveProgram::PreSwap` with a minimal `input_swap_amoun` and `input_fee_amount` set to zero.
4. Add a `JupiterV4::NopInstruction`, such as a `getter` instruction.
5. Finally, call `DefinitiveProgram::PostSwap` with `min_out` set to one.

In this sequence, both `CPIs` will use `current_index = 1` and `pre_or_post_idx = 3` in `swap_introspection_checks`, causing them to refer to the same `PostSwap` instruction. The second `CPI` overwrites the state set by the first, allowing the malicious trader to withdraw all the funds improperly.

### Remediation

To prevent this bypass, ensure the program checks it was called through `CPI`.

### Patch

Fixed in 978d1d3.

## Locked Deposited Funds HIGH OS-STP-ADV-01

### Description

The `deposit` instruction allows users to deposit tokens into a trading group using the `group_token_account`. This `group_token_account` is not constrained to be an associated token account. It may be any token account address controlled by the group. However, `post_swap` and `pre_swap` require the `group_token_account` to be an associated token account for the specific token mint involved in the swap. As a result, the deposited tokens may be locked and unusable until they are withdrawn and deposited again into an associated token account.

### Remediation

Modify the `deposit` instruction to enforce that the `group_token_account` must be an associated token account for the specific token mint allowed for deposits.

### Patch

Fixed in 1c79d08.

## Bypassing Output Token Account Check HIGH OS-STP-ADV-02

### Description

The current implementation of `swap_introspection_checks` relies on the assumption that the `output_group_token_account` should be explicitly listed within the accounts list of either the `PreSwap` or `PostSwap` instruction. However, an attacker may bypass this check by placing the `output_group_token_account` in the `remaining_accounts` list of the transaction. These `remaining_accounts` are not explicitly processed within the `swap_introspection_checks`.

>_ *src/utils/swap.rs* rust

```rust
pub fn swap_introspection_checks(
    instruction_sysvar_account_info: &AccountInfo,
    output_group_token_account: Pubkey,
) -> Result<()> {
    [...]
    // pre or post swap instruction checks (depends on who called)
    if let Ok(ix) = load_instruction_at_checked(pre_or_post_idx, &ixs) {
        [...]
        // check that output_group_token_account addresses is in matching ix
        require!(
            ix.accounts
                .iter()
                .any(|x| x.pubkey.eq(&output_group_token_account)),
            TradingError::UnknownInstruction
        );
    }
    [...]
}
```

Similarly, `postswap_matches_output_token` ensures the `PostSwap` instruction within a swap transaction references the intended `output_token_account`. Here, an attacker may also craft a malicious `PostSwap` instruction that omits the `output_token_account` from its accounts list, potentially manipulating the swap logic.

### Remediation

Ensure to validate the `remaining_accounts` list.

### Patch

Fixed in 850401e.

## Front-Running Risk In Pre-Swap Check HIGH OS-STP-ADV-03

### Description

The current implementation of the `pre_swap` instruction in the provided code is vulnerable to front-running attacks due to the method it validates the output balance. When a legitimate user initiates a `pre_swap` transaction, the attacker may submit their own transaction with the same `output_group_token_account` with a negligible amount (dust) front running the user's transaction.

Consequently, when the user's transaction performs the `output_group_token_account.amount != preswap_output_balance` check, it will fail since the attacker's deposit has altered the balance. Thus, a legitimate user's swap becomes blocked due to a failed `pre_swap` instruction.

>_ *src/handlers/handle_preswap.rs* rust

```rust
pub fn process(
    ctx: Context<PreSwap>,
    input_swap_amount: u64,
    input_fee_amount: u64,
    preswap_output_balance: u64,
) -> Result<()> {
    [...]
    // check output balance is correct
    if output_group_token_account.amount != preswap_output_balance {
        msg!(
            "preswap_output_balance ({}) did not match current amount ({})",
            preswap_output_balance,
            output_group_token_account.amount,
        );
        return err!(TradingError::IncorrectOutputBalance);
    }
    [...]
}
```

### Remediation

Instead of relying solely on the `preswap_output_balance` the user provides, the program should integrate with an Oracle price feed to verify the expected output amount based on current market conditions.

### Patch

Fixed in f9e6596.

## Potential Owner Lockout MEDIUM OS-STP-ADV-04

### Description

`RemoveGroupUser` instruction allows the removal of the group owner from the user list. However, the provided code snippet for `AddGroupUser` does not contain functionality for adding a user with the `Owner` role. This implies that once the `Owner` is removed, there is no apparent method of re-adding them with `Owner` privileges.

*>_ src/handlers/handle_remove_user.rs* rust

```rust
pub fn process(ctx: Context<RemoveGroupUser>, user: Pubkey) -> Result<()> {
    let mut group = ctx.accounts.group.load_mut()?;
    group.remove_user(user)?;
    Ok(())
}
```

### Remediation

Introduce a separate instruction for adding a new `owner` to a group.

### Patch

Fixed in 6c7043b.

## Missing Signer Check LOW OS-STP-ADV-05

### Description

`update` in `MetaConfig` allows the updating of the `definitive_admin` public key without verifying if the new `definitive_admin` public key corresponds to a signer of the current transaction. Consequently, if an admin updates the `definitive_admin` field with a public key that is not a transaction signer, the update will still succeed, potentially locking themselves out of the protocol.

*>_ src/state/meta_config.rs* rust

```rust
pub fn update(&mut self, mode: UpdateMetaConfigMode, value: &[u8]) {
    match mode {
        [...]
        UpdateMetaConfigMode::UpdateDefinitiveAdmin => {
            let new: [u8; 32] = value[0..32].try_into().unwrap();
            let new = Pubkey::new_from_array(new)
            msg!("updating admin: {} -> {}", self.definitive_admin, new);
            self.definitive_admin = new;
        }
    }
}
```

### Remediation

Compare the new `definitive_admin` public key from the value argument, within `update`, after extracting it, against the signers of the current transaction context.

### Patch

Fixed in 98929b0.

## Risk Of Fund Withdrawal By Admin LOW OS-STP-ADV-06

### Description

In the current implementation, there is a possible centralization risk as the Definitive admin has the authority to add themselves as an admin to any trading group. Since there is no restriction on the Definitive admin withdrawing funds from any group, this grants the Definitive admins unrestricted control over all trading groups, enabling them to drain funds from any group by withdrawing them without any authorization.

### Remediation

Implement proper access control for withdrawals. Only designated roles within a group should have the authority to withdraw funds.

### Patch

Fixed in 4d86972.

## Test ID In Production Code LOW

OS-STP-ADV-07

### Description

`TEST_SWAP_ID` refers to the public key used for testing swap functionalities. However, in the current implementation, it is also utilized in the production environment, which may result in unexpected behavior and security risks.

>_ *src/utils/swap.rs* rust

```rust
pub fn swap_introspection_checks(
    instruction_sysvar_account_info: &AccountInfo,
    output_group_token_account: Pubkey,
) -> Result<()> {
    [...]
    // swap instruction checks
    if let Ok(ix) = load_instruction_at_checked(swap_idx, &ixs) {
        // only accept swaps on Jupiter or out test swap program
        if !ix.program_id.eq(&JUPITER_ID) && !ix.program_id.eq(&TEST_SWAP_ID) {
            return err!(TradingError::InvalidSwap);
        }
    } else {
        return err!(TradingError::MissingExpectedInstruction);
    }
    [...]
    Ok(())
}
```

### Remediation

Utilize a compiler flag to wrap the code that includes `TEST_SWAP_ID`. This ensures the code using the test program ID is only compiled when the `test` flag is provided during compilation.

### Patch

Fixed in 845fe60.

# 05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

| ID | Description |
| --- | --- |
| OS-STP-SUG-00 | An admin set excessive fees in swap instructions. |

## Excessive Fees

OS-STP-SUG-00

### Description

Currently, the `definitive_admin` may arbitrary define `input_fee_amount` and `output_fee_bps` values during swap initiation ( `handle_preswap` ) and finalization ( `handle_postswap` ). By setting these values excessively high, the admin may drain more funds from the group's wallet than intended for legitimate swap fees.

### Remediation

Enforce reasonable maximum values for `input_fee_amount` and `output_fee_bps`.

# A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

**CRITICAL**

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

**HIGH**

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

**MEDIUM**

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

**LOW**

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

**INFO**

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

# B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.