1inch Solana Fusion

Security Assessment

June 20th, 2025 — Prepared by OtterSec

Akash Gurugunti	sud0u53r.ak@osec.io
Tamta Topuria	tamta@osec.io
Robert Chen	r@osec.io

Table of Contents

01 — Executive Summary

Overview

1inch engaged OtterSec to assess the `solana-fusion` program. This assessment was conducted between May 8th and June 19th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 4 findings throughout this audit engagement.

In particular, we identified a vulnerability, that allows an attacker to front-run the account creation instruction by preemptively initializing account, blocking the legitimate create instruction (OS-OSF-ADV-00). Furthermore, the maker may close their maker destination ATA after each fill, forcing takers to recreate it, which adds cost and disrupts future fills (OS-OSF-ADV-01).

We also recommended modifying the codebase for improved efficiency (OS-OSF-SUG-00), and advised to implement account checks for protocol destination account and integrator destination account (OS-OSF-SUG-01).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/1inch/solana-fusion-protocol. This audit was performed against commit 6879dea.

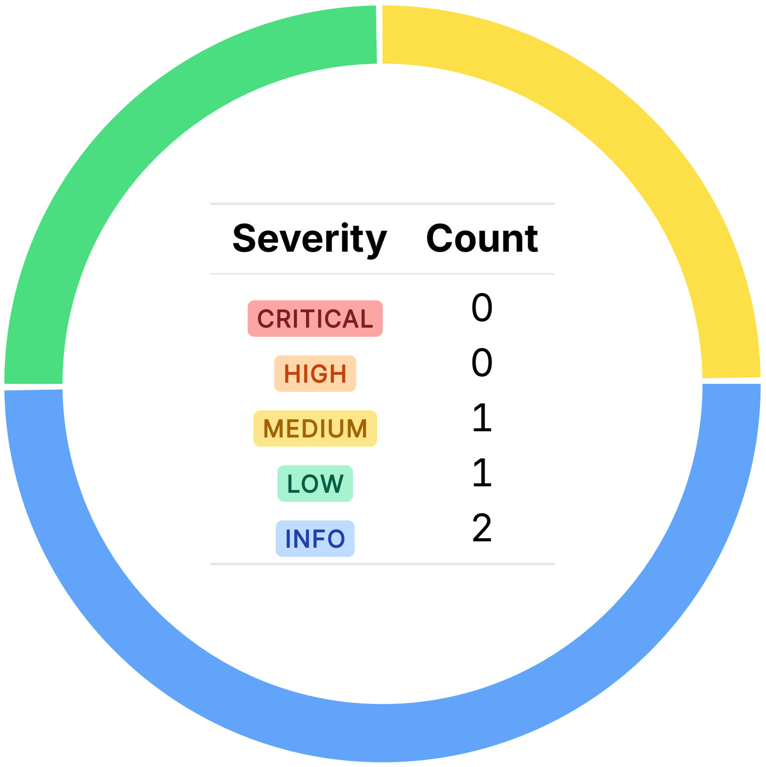
A brief description of the program is as follows:

Name	Description
solana-fusion	Enables front-running-resistant token swaps using a Dutch auction model, where professional market makers (resolvers) compete to fill user orders over time.

03 — Findings

Overall, we reported 4 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	0
HIGH	0
MEDIUM	1
LOW	1
INFO	2

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-OSF-ADV-00	MEDIUM	RESOLVED ✓	An attacker may front-run the `create` instruction by initializing the `escrow_src_ata` beforehand, failing a legitimate `create` instruction.
OS-OSF-ADV-01	LOW	RESOLVED ✓	The maker can close their `maker_dst_ata` after each fill, forcing takers to recreate it, which adds cost and disrupts future fills.

Front-Running Account Initialization MEDIUM OS-OSF-ADV-00

Description

The `escrow_src_ata` account in the `create` instruction is initialized utilizing `#[account(init)]`, which fails if the associated token account already exists. Since the address of this ATA is deterministic (based on the escrow PDA and `src_mint`), an attacker may front-run the transaction and preemptively initialize the same ATA, failing the legitimate create instruction with an `AccountAlreadyInitialized` error, resulting in a denial-of-service against the maker.

>_ programs/fusion-swap/src/lib.rs RUST

```rust
#[derive(Accounts)]
#[instruction(order: OrderConfig)]
pub struct Create<'info> {
    [...]
    /// ATA of src_mint to store escrowed tokens
    #[account(
        init,
        payer = maker,
        associated_token::mint = src_mint,
        associated_token::authority = escrow,
        associated_token::token_program = src_token_program,
    )]
    escrow_src_ata: Box<InterfaceAccount<'info, TokenAccount>>,
    [...]
}
```

Remediation

Replace `init` with `init_if_needed`.

Patch

This issue was acknowledged by the developers.

ATA Closure via Maker Post Fill LOW OS-OSF-ADV-01

Description

In the `fill` instruction, the maker may close their `maker_dst_ata` (associated token account) after each fill to reclaim rent. This breaks future fills, as the taker expects the account to exist and must recreate it before proceeding. Recreating the ATA costs extra SOL and adds complexity. The issue arises from relying on user-managed accounts without enforcing their persistence.

Remediation

Utilize an escrow account controlled by the program as the swap destination. Allow the maker to claim funds from this escrow via a separate claim instruction.

Patch

This issue was acknowledged by the developers.

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-OSF-SUG-00	Recommendation for updating the codebase to improve functionality and mitigate potential security issues.
OS-OSF-SUG-01	FusionSwap's `Create` flow lacks validation on `protocol_dst_acc` and `integrator_dst_acc`, allowing users to redirect fees to arbitrary accounts and steal protocol or integrator fees.

Code Refactoring OS-OSF-SUG-00

Description

1. `transfer_ownership` currently allows the owner to transfer control by specifying a `Pubkey`, without requiring the `new_owner` to sign. This creates a risk where a typo or incorrect key may irreversibly lock the program. Requiring the new owner to sign the transaction will ensure they control the address and consent to the transfer. This prevents accidental misassignments.
2. If a user sets the `expiration_time` earlier than the auction's end, the order may expire before resolvers access the final `PointAndTimeDelta` prices, breaking fair price discovery.

Remediation

Incorporate the above-stated refactors.

Fee Redirection Due to Improper Validation OS-OSF-SUG-01

Description

In the `Create` structure of `FusionSwap`, `protocol_dst_acc` and `integrator_dst_acc` are optional and unchecked, with no validation to ensure they belong to the protocol or a legitimate integrator. This allows a user to supply their own accounts and redirect the fees to themselves, even when `protocol_fee` and `integrator_fee` are set. However, such orders are filtered out by the backend, so resolvers using the `1inch` UI will not see them.

>_ *programs/fusion-swap/src/lib.rs* RUST

```rust
#[derive(Accounts)]
#[instruction(order: OrderConfig)]
pub struct Create<'info> {
    [...]
    protocol_dst_acc: Option<UncheckedAccount<'info>>,

    integrator_dst_acc: Option<UncheckedAccount<'info>>,
}
```

Remediation

Implement account checks for `protocol_dst_acc` and `integrator_dst_acc`.

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 others may have 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.