OFFSIDE LABS

1inch Solana Fusion

Smart Contract Security Assessment

May 2025

Prepared for:

1inch

Prepared by:

Offside Labs

Ripples Wen

Siji Feng

Contents

1 About Offside Labs **2**

2 Executive Summary **3**

3 Summary of Findings **4**

4 Key Findings and Recommendations **5**

4.1 Missing mut Annotation on maker_receiver Account in Fill Instruction . . . 5

4.2 Unpredictable ATA Rent Costs Due to Maker Front-Running . . . 6

4.3 Lack of Token-2022 Extension Handling May Cause Swap Inconsistencies . . . 7

4.4 Honeypot Dutch Auction Due to Unvalidated AuctionData . . . 8

4.5 Order State Ambiguity Due to Over-Reliance on Escrow ATA Existence . . . 9

4.6 Informational and Undetermined Issues . . . 10

5 Disclaimer **12**

1 About Offside Labs

Offside Labs stands as a pre-eminent security research team, comprising highly skilled hackers with top - tier talent from both academia and industry.

The team demonstrates extensive and diverse expertise in modern software systems, which encompasses yet are not restricted to *browsers*, *operating systems*, *IoT devices*, and *hypervisors*. Offside Labs is at the forefront of innovative domains such as *cryptocurrencies* and *blockchain technologies*. The team achieved notable accomplishments including the successful execution of remote jailbreaks on devices like the **iPhone** and **PlayStation 4**, as well as the identification and resolution of critical vulnerabilities within the **Tron Network**.

Offside Labs actively involves in and keeps contributing to the security community. The team was the winner and co-organizer for the *DEFCON CTF*, the most renowned CTF competition in Web2. The team also triumphed in the **Paradigm CTF 2023** in Web3. Meanwhile, the team has been conducting responsible disclosure of numerous vulnerabilities to leading technology companies, including *Apple*, *Google*, and *Microsoft*, safeguarding digital assets with an estimated value exceeding **$300 million**.

During the transition to Web3, Offside Labs has attained remarkable success. The team has earned over **$9 million** in bug bounties, and **three** of its innovative techniques were acknowledged as being among the **top 10 blockchain hacking techniques of 2022** by the Web3 security community.

2 Executive Summary

Introduction

Offside Labs completed a security audit of *1inch Solana Fusion* smart contracts, starting on April 7th, 2025, and concluding on April 16th, 2025.

Project Overview

This project develops a limit order trading system on the Solana blockchain with support for Dutch auctions. In this system:

- Dutch Auction Mechanism: Order prices gradually decrease over time.
- Order Execution: The auction continues until someone places a bid or the order expires.
- Free Cancellation: Users can cancel their orders for free at any time.
- Timeout Cancellation Fee: A fee is incurred only when an order is canceled by another party after the order has reached its timeout period.

Audit Scope

The assessment scope contains mainly the smart contracts of the fusion-swap program for the *1inch Solana Fusion* project.

The audit is based on the following specific branches and commit hashes of the codebase repositories:

- 1inch Solana Fusion
  - Codebase: `https://github.com/1inch/solana-fusion-protocol`
  - Commit Hash: bffb014892970acb6e1e3b531bb90761752289cf

We listed the files we have audited below:

- 1inch Solana Fusion
  - programs/fusion-swap/src/*.rs
  - programs/whitelist/src/*.rs

Findings

The security audit revealed:

- 0 critical issue
- 0 high issue
- 2 medium issues
- 3 low issues
- 4 informational issues

Further details, including the nature of these issues and recommendations for their remediation, are detailed in the subsequent sections of this report.

3 Summary of Findings

ID	Title	Severity	Status
01	Missing mut Annotation on maker_receiver Account in Fill Instruction	Medium	Fixed
02	Unpredictable ATA Rent Costs Due to Maker Front-Running	Medium	Acknowledged
03	Lack of Token-2022 Extension Handling May Cause Swap Inconsistencies	Low	Acknowledged
04	Honeypot Dutch Auction Due to Unvalidated AuctionData	Low	Acknowledged
05	Order State Ambiguity Due to Over-Reliance on Escrow ATA Existence	Low	Acknowledged
06	Fill Instruction Fails When Fee Recipients Are Below Rent-Exempt Threshold	Informational	Acknowledged
07	Lack of Fee Limit Checks Leads to Overflow in get_fee_amounts	Informational	Acknowledged
08	Unconditional init on escrow_src_ata Enables Front-Running DoS	Informational	Acknowledged
09	Missing Event Emission for Order Actions	Informational	Acknowledged

4 Key Findings and Recommendations

4.1 Missing mut Annotation on maker_receiver Account in Fill Instruction

Severity: Medium	Status: Fixed
Target: Smart Contract	Category: Logic Error

Description

```
/// CHECK: maker_receiver only has to be equal to escrow parameter
maker_receiver: UncheckedAccount<'info>,
```

programs/fusion-swap/src/lib.rs#L493-L494

```
// Taker => Maker
let mut params = if order.dst_asset_is_native {
    UniTransferParams::NativeTransfer {
        from: ctx.accounts.taker.to_account_info(),
        to: ctx.accounts.maker_receiver.to_account_info(),
        amount: maker_dst_amount,
        program: ctx.accounts.system_program.clone(),
    }
} else {
```

programs/fusion-swap/src/lib.rs#L192-L199

In the `Fill` instruction, the `maker_receiver` account is intended to receive lamports when `dst_asset_is_native` is true. However, this account lacks the `mut` annotation, which is necessary to allow the account's balance to be modified

Impact

The absence of the `mut` annotation on the `maker_receiver` account will cause the transaction to fail when the logic attempts to transfer lamports to this account (when `dst_asset_is_native` is true). It's important to note that if the `maker` account and the `maker_receiver` account happen to be the same, the transaction will not fail because the `maker` account is already correctly annotated with `mut`. This creates an inconsistent and potentially confusing behavior.

Recommendation

Add the missing `mut` annotation to the `maker_receiver` account in the `Fill` instruction definition. This will allow the account's balance to be updated, enabling it to correctly receive lamports when `dst_asset_is_native` is true, and resolve the inconsistency.

Mitigation Review Log

Fixed at commit 0b4753181a55a1efa7e813b594105bbe53ea1405.

4.2 Unpredictable ATA Rent Costs Due to Maker Front-Running

Severity: Medium	**Status: Acknowledged**
Target: Smart Contract	**Category: Logic Error**

Description

The protocol requires takers to pay for initializing the maker's destination ATA during order fulfillment using `init_if_needed` . However, makers can front-run the taker's transaction by closing their ATA after simulation but before execution. This forces takers to unexpectedly cover the ATA rent cost, particularly impactful for small orders where rent becomes a significant percentage of the trade value.

```
    /// Maker's ATA of dst_mint
    #[account(
        init_if_needed,
        payer = taker,
        associated_token::mint = dst_mint,
        associated_token::authority = maker_receiver,
        associated_token::token_program = dst_token_program,
    )]
    maker_dst_ata: Option<Box<InterfaceAccount<'info, TokenAccount>>>,
```

programs/fusion-swap/src/lib.rs#L541-L549

Impact

Takers face unpredictable cost increases, while makers can exploit this to extract value. The protocol's simulation results become unreliable, and the economic burden shifts unfairly to takers.

Recommendation

Initialize the maker's ATA during order creation instead of fulfillment, requiring makers to pay rent upfront. This guarantees the ATA exists for the order's duration, eliminates rent uncertainty for takers, and removes the exploit while maintaining storage efficiency.

4.3 Lack of Token-2022 Extension Handling May Cause Swap Inconsistencies

Severity: Low	**Status: Acknowledged**
Target: Smart Contract	**Category: Logic Error**

Description

The current implementation of the swap contract utilizes types from `anchor_spl::token_interface` for handling token-related accounts, indicating support for the standard SPL Token program and by extension, Token-2022. However, the contract logic does not explicitly account for certain Token-2022 extensions, particularly the transfer fee extension and the transfer hook extension.

Impact

The lack of consideration for the transfer fee extension can lead to a discrepancy between the expected and actual amount of tokens received by a user during a swap. If a swapped token has the transfer fee extension enabled, a portion of the transferred amount will be deducted as a fee. Since the contract doesn't factor this fee into its calculations, users might receive less of the desired token than anticipated based on the swap rate and the amount they initially provided. This can result in financial loss and a negative user experience.

Similarly, the transfer hook extension introduces programmable logic that can be executed during transfers. If not properly handled, this can lead to several critical issues like failed transactions, unexpected state changes, security vulnerabilities and incorrect balance updates.

These risks can compromise the integrity and reliability of the swap contract.

Recommendation

To ensure robust support for Token-2022 tokens, the swap contract should detect the presence of Token-2022 extensions, especially the transfer fee and transfer hook extensions, and implement logic to handle them appropriately, ensuring accurate accounting and safe execution of transfers.

If support for such extensions is not intended, the contract should proactively reject tokens that include unsupported extensions during the order creation or initialization phase. This validation should be enforced on-chain to prevent misuse and ensure predictable behavior.

4.4 Honeypot Dutch Auction Due to Unvalidated AuctionData

Severity: Low	Status: Acknowledged
Target: Smart Contract	Category: Logic Error

Description

The `Create` instruction accepts unvalidated `dutch_auction_data`, allowing malicious makers to craft deceptive orders where:

1. Arbitrarily small `time_delta`: The time interval between price drops can be set to extremely short durations (e.g., 1 second), enabling rapid price manipulation.
2. Non-decreasing rate bumps: The initial rate bump can start at 0%, then spike to an exorbitant value in the next interval.

```rust
    let mut current_rate_bump = data.initial_rate_bump as u64;
    let mut current_point_time = data.start_time as u64;

    for point_and_time_delta in data.points_and_time_deltas.iter() {
        let next_rate_bump = point_and_time_delta.rate_bump as u64;
        let point_time_delta = point_and_time_delta.time_delta as u64;
        let next_point_time = current_point_time + point_time_delta;

        if timestamp <= next_point_time {
            // Overflow is not possible because:
            // 1. current_point_time < timestamp <= next_point_time
            // 2. size(timestamp) + size(rate_bump) < 64
            // 3. point_time_delta != 0 as this would contradict point 1
            return ((timestamp - current_point_time) * next_rate_bump
                + (next_point_time - timestamp) * current_rate_bump)
                / point_time_delta;
        }

        current_rate_bump = next_rate_bump;
        current_point_time = next_point_time;
    }
```

programs/fusion-swap/src/auction.rs#L26-L46

During order fulfillment, the Fill instruction calculates the current price based on the on-chain timestamp, making it possible for an attacker to:

1. Lure takers with a seemingly favorable initial rate.
2. Trigger a sudden rate increase after the taker simulates the transaction, forcing them to overpay.

Impact

A taker expecting a fair auction may unknowingly agree to pay a massively inflated price due to timestamp-dependent rate spikes. Since the rate depends on the actual execution timestamp, local simulations may fail to detect the trap.

Unlike EVM's ERC20 approvals, Solana's token program lacks a spending limit mechanism. Once a taker signs, the program can withdraw unlimited tokens from their account, exacerbating the risk.

Proof of Concept

1. Attacker creates a malicious order:
   - Sets `initial_rate_bump = 0` and `points_and_time_deltas = vec![(1000000, 1)]` (0% initial rate, jumping to 1000% after 1 second).
   - Lists an attractive initial price to lure takers.
2. Taker signs the `Fill` transaction:
   - At signing time (`t=0`), the price appears fair (0% bump).
   - Due to network latency, execution occurs at `t=1` , activating the 1000% rate bump.
3. Taker overpays:
   - The program calculates the price at `t=1` and charges the taker **10x the expected amount**.

The bump rate changes based on the on-chain timestamp. An attacker can increase their deception success rate by rapidly "flipping" the bump rate - alternating between zero and an extremely high value using a long `Vec<PointAndTimeDelta>` sequence.

Recommendation

1. Enforce rate monotonicity: Modify the auction logic to ensure each `rate_bump` is strictly less than the previous one, e.g. `current_rate_bump = current_rate_bump - next_rate_bump_diff`
2. Add taker spending limits: Introduce a parameter in `Fill` (e.g., `max_dst_amount`) to revert if the calculated price exceeds the taker's specified limit.
3. Minimum time_delta constraint: Require `time_delta >= MIN_INTERVAL` (e.g., 5 minutes) to prevent micro-manipulation.

4.5 Order State Ambiguity Due to Over-Reliance on Escrow ATA Existence

Severity: Low	Status: Acknowledged
Target: Smart Contract	Category: Design Flaw

Description

The protocol currently uses the existence of the ATA of the escrow as the sole indicator of an active order. While this minimizes storage costs, it introduces some flaws:

1. No enforced state transitions: Once an order is canceled (ATA closed), anyone can forcibly "revive" it by manually recreating the ATA.
2. Unintended reuse: Even the original maker can accidentally reuse a canceled order, creating confusion for takers and indexers.

Since the escrow ATA is not fully controlled by the program, its state can be altered externally, breaking the expected order lifecycle.

Impact

- A taker may fill an order they believe is active, only to later discover it was canceled and revived without the maker's intent.
- Makers lose guarantees about order finality, undermining trust in cancellation.
- Off-chain systems cannot reliably distinguish between legitimately renewed orders and artificially revived ones.

Recommendation

To resolve this issue, we could initialize an on-chain storage account for each order – though this may seem counterintuitive to the current minimal design.

A critical distinction between Solana and EVM is their storage cost model:

- On **Solana**, the rent paid for account storage is **fully reclaimable** when the account is closed.
- On **EVM**, gas costs for storage provide only **partial refunds** when data is cleared.

While the upfront rent cost appears non-trivial, it's ultimately temporary and recoverable. By initializing a dedicated escrow account, we can securely store key order information (such as completion status and filled amounts) on-chain, eliminating ambiguity while maintaining financial efficiency.

4.6 Informational and Undetermined Issues

Fill Instruction Fails When Fee Recipients Are Below Rent-Exempt Threshold

Severity: Informational	Status: Acknowledged
Target: Smart Contract	Category: Logic Error

Within the `Fill` instruction, the `protocol_dst_acc` and `integrator_dst_acc` are intended recipients of fees. If either of these accounts has a zero balance (or a balance below the minimum rent-exempt amount) and the calculated fee to be transferred is less than the lamports required to make the account rent-exempt, the `system_instruction::transfer` will fail. This is a fundamental constraint of the Solana runtime, which prevents accounts from falling below the rent-exempt threshold. Consequently, the entire `Fill` transaction will fail, even if the underlying trade was successfully executed.

Lack of Fee Limit Checks Leads to Overflow in get_fee_amounts

Severity: Informational	**Status: Acknowledged**
Target: Smart Contract	**Category: Missing Validation**

During order creation, users or integrators can specify `protocol_fee` and `integrator_fee`. However, there is no validation to ensure that their combined value remains within a reasonable and processable limit. If the sum of these fees exceeds `BASE_1E5`, the subsequent calculation within `get_fee_amounts` will overflow, leading to a transaction failure during the fill process.

Unconditional init on escrow_src_ata Enables Front-Running DoS

Severity: Informational	**Status: Acknowledged**
Target: Smart Contract	**Category: Logic Error**

In the `Create` instruction, the `escrow_src_ata` account is set to auto-initialize using Anchor's `init` attribute. If this account already exists when `Create` runs, the transaction will fail due to a re-initialization error. An attacker could monitor pending `Create` transactions and pre-create the `escrow_src_ata` account, causing the legitimate transaction to fail. While this doesn't directly profit the attacker, it can disrupt the protocol by blocking order creation.

Missing Event Emission for Order Actions

Severity: Informational	**Status: Acknowledged**
Target: Smart Contract	**Category: Design Flaw**

The program currently lacks structured event logging, making it difficult to track on-chain activity (e.g., order creation, fills, or cancellations) efficiently. Best practices recommend emitting events either via Anchor's `#[event]` macro (for simplicity) or a dedicated library like `event-cpi` (for schema validation and CPI-based logging). Implementing events would improve off-chain indexing, debugging, and transparency without requiring storage overhead.

5 Disclaimer

This report reflects the security status of the project as of the date of the audit. It is intended solely for informational purposes and should not be used as investment advice. Despite carrying out a comprehensive review and analysis of the relevant smart contracts, it is important to note that Offside Labs' services do not encompass an exhaustive security assessment. The primary objective of the audit is to identify potential security vulnerabilities to the best of the team's ability; however, this audit does not guarantee that the project is entirely immune to future risks.

Offside Labs disclaims any liability for losses or damages resulting from the use of this report or from any future security breaches. The team strongly recommends that clients undertake multiple independent audits and implement a public bug bounty program to enhance the security of their smart contracts.

The audit is limited to the specific areas defined in Offside Labs' engagement and does not cover all potential risks or vulnerabilities. Security is an ongoing process, regular audits and monitoring are advised.

Please note: Offside Labs is not responsible for security issues stemming from developer errors or misconfigurations during contract deployment and does not assume liability for centralized governance risks within the project. The team is not accountable for any impact on the project's security or availability due to significant damage to the underlying blockchain infrastructure.

By utilizing this report, the client acknowledges the inherent limitations of the audit process and agrees that the firm shall not be held liable for any incidents that may occur after the completion of this audit.

This report should be considered null and void in case of any alteration.

OFFSIDE LABS

https://offside.io/

https://github.com/offsidelabs

https://twitter.com/offside_labs