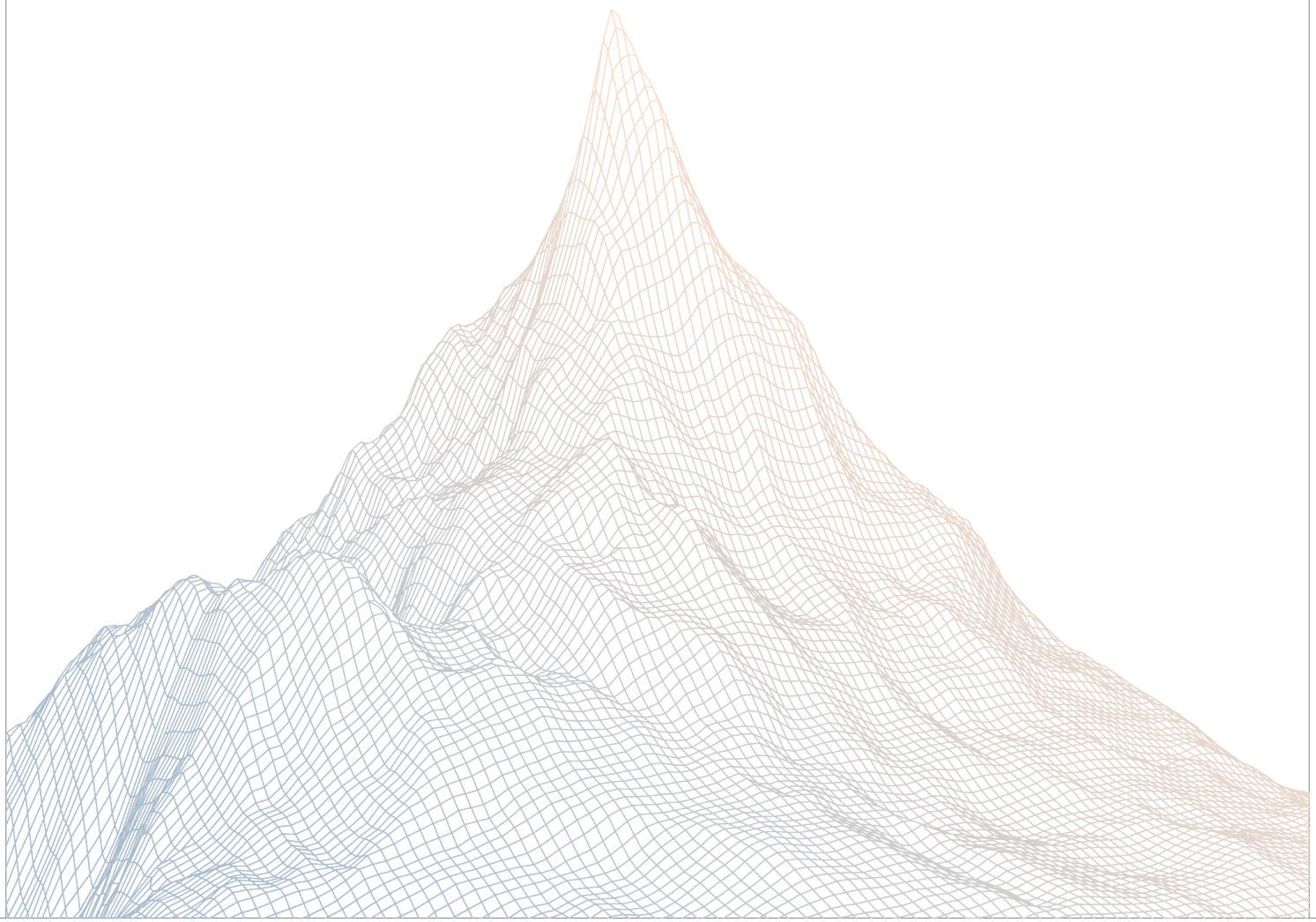
Zenith

1inch

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: June 25th to June 26th, 2025

AUDITED BY: Matte
Peakbolt

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About 1inch

The 1inch Fusion+ API is a powerful solution for secure and efficient cross-chain swaps in DeFi that uses a creative architecture of Dutch auctions and automated recovery, all without relying on a single centralized custodian.

2.2 Scope

The engagement involved a review of the following targets:

Target	solana-crosschain-protocol
Repository	https://github.com/1inch/solana-crosschain-protocol/
Commit Hash	`f6bfbe46593e5d42e888511829436c4511d1ee90`

Files	`cross-chain-escrow-dst/*`
	`cross-chain-escrow-src/*`
	`whitelist/*`

Target	solana-crosschain-protocol
Repository	https://github.com/1inch/solana-crosschain-protocol/
Commit Hash	`06ef2533ab5dd451a6d61bda677d54e6e628e21e`
Files	`programs/**/src/*.rs` `common/src/*.rs`

2.3 Audit Timeline

June 25, 2025	Audit start
June 26, 2025	Audit end
July 16, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	2
Medium Risk	2
Low Risk	1
Informational	1
Total Issues	**6**

3

Findings Summary

ID	Description	Status
H-1	Preemptive creation of escrow_ata and order_ata will DoS cross-chain swap	Resolved
H-2	Incorrect amount and dst_amount used for EscrowSrc will lead to erroneous swap	Resolved
M-1	Token2022 Extension Mishandling	Acknowledged
M-2	A significantly large parts_amount will cause silent overflow in is_valid_partial_fill() leading to an incorrect validation	Resolved
L-1	Maker can block escrowed token transfer in cancel_escrow() and withdraw() to earn safety deposit	Acknowledged
I-1	Store and load bumps for Order, EscrowSrc and EscrowDst PDA	Resolved

4

Findings

4.1 High Risk

A total of 2 high risk findings were identified.

[H-1] Preemptive creation of `escrow_ata` and `order_ata` will DoS cross-chain swap

SEVERITY: High	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

- cross-chain-escrow-src/src/lib.rs#L687-L695
- cross-chain-escrow-dst/src/lib.rs#L227-L235
- cross-chain-escrow-src/src/lib.rs#L603-L611

Description:

The instructions for creating `Order`, `EscrowSrc` and `EscrowDst` PDAs will also init the corresponding ATAs to store the escrowed tokens.

However, as these ATAs can be created outside of the fusion+ program, it is possible for one to preemptively create them using known PDA seeds and cause the creation of Order/EscrowSrc/EscrowDst to fail.

This will effectively allow an attacker to DoS the fusion+ cross-chain swap by preventing the maker/taker from creating and filling orders.

```
pub struct CreateEscrow<'info> {
    ...
    #[account(
        init,
        payer = taker,
        associated_token::mint = mint,
        associated_token::authority = escrow,
        associated_token::token_program = token_program
    )]
    escrow_ata: Box<InterfaceAccount<'info, TokenAccount>>,
```

```
pub struct Create<'info> {
    ...
    #[account(
        init,
        payer = creator,
        associated_token::mint = mint,
        associated_token::authority = escrow,
        associated_token::token_program = token_program
    )]
    escrow_ata: Box<InterfaceAccount<'info, TokenAccount>>,
```

```
pub struct Create<'info> {
    ...
    #[account(
        init,
        payer = creator,
        associated_token::mint = mint,
        associated_token::authority = order,
        associated_token::token_program = token_program
    )]
    order_ata: Box<InterfaceAccount<'info, TokenAccount>>,
```

Recommendations:

Use `init_if_needed` for the `order_ata` and `escrow_ata` instead.

1inch: Resolved with @86e2c2883...

Zenith: Verified.

[H-2] Incorrect `amount` and `dst_amount` used for `EscrowSrc` will lead to erroneous swap

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- cross-chain-escrow-src/src/lib.rs#L203-L218

Description:

The taker calls `cross_chain_escrow_src::create_escrow()` to transfer the maker's input amount from the `order_ata` to the `escrow_ata`. It then records the `escrow.amount` and `escrow.dst_amount` as shown in the code below.

However, it incorrectly uses `order.amount` instead of the actual making `amount`.

This will cause the wrong amount to be used during `withdraw()` and will fail for a partial fill as the amount be transferred to the taker will be more than the balance of `escrow_ata`. This prevents taker from withdrawing the escrow amount on the source chain.

The same issue also applies for `escrow.dst_amount`, which stores the entire `dst_amount` instead of the actual `dst_amount`, which could be lower for a partial fill. This will cause a higher dst_amount used for the destination chain, causing the maker to receive more than expected.

```
ctx.accounts.escrow.set_inner(EscrowSrc {
    order_hash: order.order_hash,
    hashlock,
    maker: order.creator,
    taker: ctx.accounts.taker.key(),
    token: order.token,
    //@audit this should be amount
    amount: order.amount,
    safety_deposit: order.safety_deposit,
    withdrawal_start,
    public_withdrawal_start,
    cancellation_start,
    public_cancellation_start,
    rescue_start: order.rescue_start,
    asset_is_native: order.asset_is_native,
```

```
    //@audit this should be derived from  the filled amount
    dst_amount: get_dst_amount(order.dst_amount, &dutch_auction_data)?,
});
```

Recommendations:

Update `cross_chain_escrow_src::create_escrow()` to store the actual `amount` and `dst_amount` that will be filled.

1inch: Resolved with @6e0ba0911... and @5595c3268...

Zenith: Verified.

4.2 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Token2022 Extension Mishandling

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- cross-chain-escrow-dst/src/lib.rs

Description:

The protocol blindly accepts Token2022 mints without accounting for their specialized behaviors, causing catastrophic failures when extension-enabled tokens interact with the swap mechanism. These enhanced tokens carry unique rules - transfer fees that shrink balances during movement, confidential transfers requiring specific unwinding procedures, and strict closure conditions that demand zero withheld amounts.

The escrow programs, built with standard SPL tokens in mind, violate these rules at every turn: they record pre-fee amounts while storing post-fee balances, attempt to close accounts with outstanding obligations, and force tokens through paths that trigger multiple fee deductions.

This fundamental incompatibility transforms routine operations into irreversible failures - orders become unmatchable due to balance mismatches, withdrawals fail when accounts refuse closure, and users hemorrhage value through compounded fees. What should be a straightforward cross-chain swap devolves into a trap where advanced token features become weapons against the protocol itself.

Recommendations:

Only allow TokenMetadata and MetadataPointer extensions

1inch: Acknowledged

[M-2] A significantly large `parts_amount` will cause silent overflow in `is_valid_partial_fill()` leading to an incorrect validation

SEVERITY: Medium	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- cross-chain-escrow-src/src/lib.rs#L1197-L1223

Description:

Taker calls `create_escrow()` to prepare the escrow ATA for order making. For partial fill orders, the taker is required to specify the parameter `parts_amount`, which indicates how many equal portions that the order is segmented into.

However, the taker could specify a significantly high value for `parts_amount`, causing it to silently overflow the `is_valid_partial_fill()` calculation. This allows the taker to be able to obtain an index that do not correctly correspond to the `making amount`.

```
fn is_valid_partial_fill(
    making_amount: u64,
    remaining_making_amount: u64,
    order_making_amount: u64,
    parts_amount: u64,
    validated_index: u64,
) → bool {
    //@audit this will silently overflow if parts_amount is significantly
    large
    let calculated_index = ((order_making_amount - remaining_making_amount +
    making_amount - 1)
        * parts_amount)
        / order_making_amount;
```

Recommendations:

Limit `parts_amount` to a reasonable sized variable such as u8.

1inch: Resolved with @a2edfa47f....

Zenith: Verified.

4.3 Low Risk

A total of 1 low risk findings were identified.

[L-1] Maker can block escrowed token transfer in `cancel_escrow()` and `withdraw()` to earn safety deposit

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- cross-chain-escrow-src/src/lib.rs#L305-L315
- cross-chain-escrow-dst/src/lib.rs#L96-L97

Description:

Both `cross_chain_src::cancel_escrow()` and `cross_chain_dst::withdraw()` can be called by the taker to transfer the escrowed tokens to the maker. If these instructions were not successfully called by the taker, the token transfer can be performed by the whitelisted resolvers using `public_cancel_escrow()` and `public_withdraw()`.

The issue is that this allows the maker to temporarily block the escrowed token transfer and DoS `cancel_escrow()` and `withdraw()` until the the resolvers can call `public_cancel_escrow()` and `public_withdraw()`. That will cause the taker to lose the safety deposit to the resolver that called those instructions.

The risk of this occurring is low as it means that the maker is a resolver (or colludes with a resolver), who is unlikely to perform this attack as resolvers are trusted parties.

```
pub fn cancel_escrow(ctx: Context<CancelEscrow>) -> Result<()> {
    ...
    // In a standard cancel, the taker receives the entire rent amount,
    including the safety deposit,
    // because they initially covered the entire rent during escrow creation,
    while the maker
    // receives their tokens back to their initial ATA or wallet if the token
    is native.

    common::escrow::cancel(
```

```
        &ctx.accounts.escrow,
        ctx.bumps.escrow,
        &ctx.accounts.escrow_ata,
        ctx.accounts.maker_ata.as_deref(), // order creator ATA
        &ctx.accounts.mint,
        &ctx.accounts.token_program,
        &ctx.accounts.taker, // rent recipient
        &ctx.accounts.maker, // order creator
        &ctx.accounts.taker, // safety deposit recipient
    )
}
```

```
pub fn withdraw(ctx: Context<Withdraw>, secret: [u8; 32]) -> Result<()> {
    ...
    // In a standard withdrawal, the creator receives the entire rent amount,
    including the safety deposit,
    // because they initially covered the entire rent during escrow creation.

    common::escrow::withdraw(
        &ctx.accounts.escrow,
        ctx.bumps.escrow,
        &ctx.accounts.escrow_ata,
        &ctx.accounts.recipient,
        ctx.accounts.recipient_ata.as_deref(),
        &ctx.accounts.mint,
        &ctx.accounts.token_program,
        &ctx.accounts.creator,
        &ctx.accounts.creator,
        secret,
    )
}
```

Recommendations:

Consider using a pull model in both `cancel_escrow()` and `withdraw()` by transferring the escrowed tokens to a program controlled account where the maker has to call an instruction to retrieve the tokens.

1inch: Acknowledged. We find it highly unlikely that KYB-verified resolvers would collude with a maker to block token transfers in order to extract a relatively small safety deposit, especially considering the reputational risks involved. If such behavior is ever reported, the resolver in question will lose their status.

4.4 Informational

A total of 1 informational findings were identified.

[I-1] Store and load bumps for `Order`, `EscrowSrc` and `EscrowDst` PDA

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- cross-chain-escrow-src/src/lib.rs
- cross-chain-escrow-dst/src/lib.rs

Recommendations:

The bumps for the `Order`, `EscrowSrc`, `EscrowDst` PDA should be stored and then loaded to save CU for searching the canonical bump.

1inch: Resolved with @b34662a31....

Zenith: Verified.