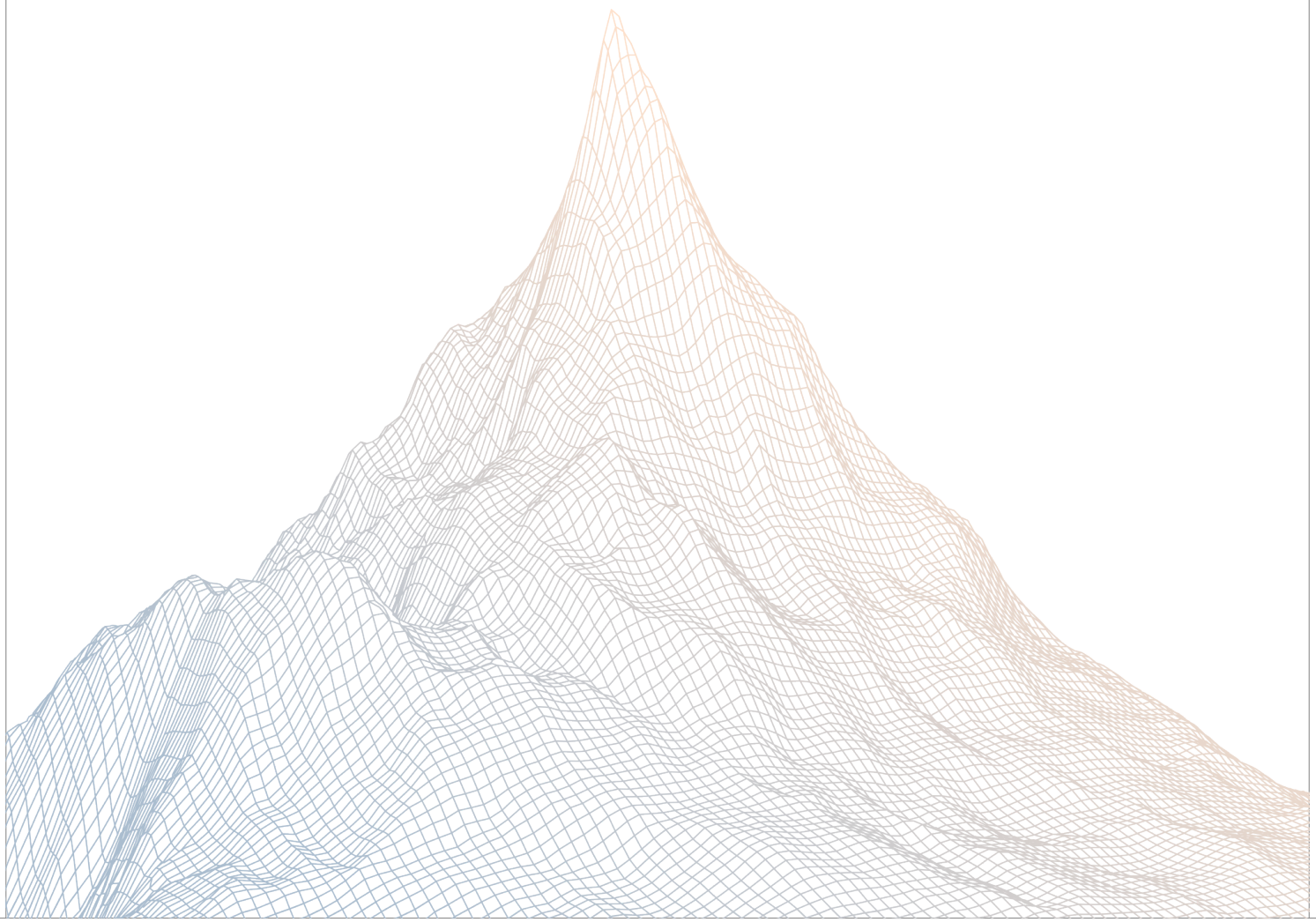
Zenith

1inch

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: March 24th to March 26th, 2025

AUDITED BY: J4X
Peakbolt

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

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-fusion-protocol
Repository	https://github.com/1inch/solana-fusion-protocol
Commit Hash	`cdc953b06aba4dc2da9c0a8da88da35b18bd0298`
Files	`programs/*`

2.3 Audit Timeline

March 24, 2025	Audit start
March 26, 2025	Audit end
April 15, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	3
Medium Risk	3
Low Risk	8
Informational	0
Total Issues	**14**

3

Findings Summary

ID	Description	Status
H-1	Missing checks on integrator_dst_acc and protocol_dst_acc	Acknowledged
H-2	Protocol fee can be bypassed	Acknowledged
H-3	Surplus fee can be bypassed by setting expected value to max	Acknowledged
M-1	Maker can force taker to pay ATA rent for him	Acknowledged
M-2	Closure of maker's escrow ATA in fill() can be disrupted by spamming dust donation	Acknowledged
M-3	create can be griefed by creating the escrow ATA ahead of it	Acknowledged
L-1	Maker can top-up escrow ATA to sell more than indicated to evade fee hikes	Acknowledged
L-2	Varying CU due to search for canonical bump could cause fill() to exceed CU limit	Resolved
L-3	Missing validation of the AuctionData	Acknowledged
L-4	Fees are rounded down	Acknowledged
L-5	Complete fill() can be DoS by spamming dust fill	Acknowledged
L-6	Fees are not verified to be less than 100%	Acknowledged
L-7	Takers can fill expired orders	Resolved
L-8	Protocol allows Token2022 extension but does not handle or reject them	Acknowledged

4

Findings

4.1 High Risk

A total of 3 high risk findings were identified.

[H-1] Missing checks on `integrator_dst_acc` and `protocol_dst_acc`

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

Target

- programs/fusion-swap/src/lib.rs#L470-L472

Description:

On creating an order the maker has to provide the `integrator_dst_acc` and `protocol_dst_acc` accounts.

```
protocol_dst_acc: Option<UncheckedAccount<'info>>,

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

The fees will be distributed to these accounts. However, there is no check that these accounts are held by the protocol/integrator. As a result, the maker can pass his accounts and not pay any fees.

Recommendations:

We recommend verifying both accounts against accounts defined by the protocol in an additional system-wide config account.

1Inch: These checks will be done by the backend. Orders with incorrect protocol or integrator accounts will not be tracked or shared with resolvers.

Zenith: Acknowledged by client due to checks being done off-chain.

[H-2] Protocol fee can be bypassed

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

Target

- programs/fusion-swap/src/lib.rs#L44

Description:

Makers use the `create` instruction to create new orders. In the instruction, the maker can provide the `OrderConfig`.

```
pub struct OrderConfig {
    id: u32,
    src_amount: u64,
    min_dst_amount: u64,
    estimated_dst_amount: u64,
    expiration_time: u32,
    src_asset_is_native: bool,
    dst_asset_is_native: bool,
    fee: FeeConfig,
    dutch_auction_data: AuctionData,
    cancellation_auction_duration: u32,
}
```

The struct includes the fee configuration, including the fee that will go to the protocol and integrators.

```
pub struct FeeConfig {
    /// Protocol fee in basis points where `BASE_1E5` = 100%
    protocol_fee: u16,

    /// Integrator fee in basis points where `BASE_1E5` = 100%
    integrator_fee: u16,

    /// Percentage of positive slippage taken by the protocol as an
    additional fee.
    /// Value in basis points where `BASE_1E2` = 100%
```

```
    surplus_percentage: u8,

    /// Maximum cancellation premium
    /// Value in absolute lamports amount
    max_cancellation_premium: u64,
}
```

However, this fee configuration is never verified, so the maker can set all fees to zero.

Recommendations:

We recommend adding an account created once by the admins and keeping track of the configured protocol fees. The user-provided fees should then be validated against this account.

1Inch: Again, the backend will check for incorrect fee parameters. Any such orders will not be tracked in the database or shared with resolvers.

Zenith: Acknowledged due to check being done off-chain.

[H-3] Surplus fee can be bypassed by setting expected value to max

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

Target

- programs/fusion-swap/src/lib.rs#L773-L777

Description:

The surplus fee is charged on the funds that the user will receive more than he has expected.

```
if actual_dst_amount > estimated_dst_amount {
    protocol_fee_amount += (actual_dst_amount - estimated_dst_amount)
        .mul_div_floor(surplus_percentage as u64, BASE_1E2)
        .ok_or(ProgramError::ArithmeticOverflow)?;
}
```

However, the user can set the expected amount. So a maker can just set this to `u64.max` and never pay any fees.

Recommendations:

We recommend basing the surplus fee on the difference between the minimum and the actual execution.

1Inch: The backend will check for such malicious fee parameters. Orders with invalid parameters will not be tracked in the database or shared with resolvers.

Zenith: Acknowledged due to check being done off-chain.

4.2 Medium Risk

A total of 3 medium risk findings were identified.

[M-1] Maker can force taker to pay ATA rent for him

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

Target

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

Description:

When a trade is fulfilled, the taker's tokens are sent to an ATA for the maker. However, the maker does not have to create this ATA in the create function (and could close it himself later even if he were forced to). As a result, the taker has to create this ATA in the `fill` IX.

```
/// 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>>>,
```

The maker can intentionally do this, as he can later close this ATA and reclaim the rent which was paid for by the taker.

Recommendations:

We recommend implementing a two-step process: The tokens are sent to an escrow created by the maker on `create`. The maker can then claim them from this escrow and create an ATA.

1inch: This scenario is equivalent to the maker selling tokens at a slightly higher price. If

the price is unfavorable, the taker can simply choose not to fill the order.

Zenith: Acknowledged by client

[M-2] Closure of maker's escrow ATA in `fill()` can be disrupted by spamming dust donation

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

Target

- programs/fusion-swap/src/lib.rs#L123

Description:

When a taker fill the order completely, the `escrow_src_ata` is closed on behalf of the maker. That occurs in `fill()` when `amount` = escrow_src_ata.amount‘.

However, it is possible for an anyone to prevent the escrow account closure by spamming dust donation to `escrow_src_ata` at the start of every slot. This will cause `escrow_src_ata.amount` to be higher than `amount`, resulting in a parital fill and leave the escrow account unclosed.

This will then require the maker to close the account using `cancel()` or pay for the premium for resolver to cancel on behalf.

Recommendations:

Consider tracking the filled amount so that a complete fill will occur and close the account regardless of any donation.

1inch: We do not create an escrow PDA to store the remaining tokens and prefer to leave the current behavior as is, since it does not disrupt the order lifecycle. This way, the maker pays lower fees when calling create.

Zenith: Acknowledged by client.

[M-3] `create` can be griefed by creating the escrow ATA ahead of it

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

Target

- programs/fusion-swap/src/lib.rs#L441

Description:

The `create()` instruction will create the `escrow_src_ata` PDA as specified by the `init` constraint.

However, `init` will return an error when the `escrow_src_ata` PDA already exists. This is possible as the ATA can be created by calling the Assocated Token Program directly, without using the fusion swap program.

That means an attacker can easily DoS the `create()` instruction by creating the `escrow_src_ata` before `create()`.

This will require the user to perform another tx to transfer the funds into escrow.

```
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>>,
```

Recommendations:

Use `init_if_needed` instead.

1inch: The proposed fix—using init_if_needed instead of init—would allow create to be

called multiple times for the same order. In our opinion, this could lead to greater risks for the user than the potential front-running of the protocol.

Zenith: Acknowledged by client.

4.3 Low Risk

A total of 8 low risk findings were identified.

[L-1] Maker can top-up escrow ATA to sell more than indicated to evade fee hikes

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

Target

- programs/fusion-swap/src/lib.rs#L123

Description:

The `OrderConfig` struct contains the maker's requested swap `src_amount` for the order, which is transferred into the escrow account on `create()`. The source amount available to be filled is determined by the `escrow_src_ata.amount`.

As `fill()` does not track the filled amount, it means the maker can change the amount to be filled by transferring into `escrow_src_ata.amount`. This allows maker to sell more tokens than was indicated in the order, potentially messing up any frontend calculation.

Makers can also abuse this where there is a fee hike (for protocol/integrator fee), by topping up the order with the old `FeeConfig` to benefit as much as possible from the lower fee. Though this abuse has a limited time window as the auction price will drop or expire over time.

```
pub struct OrderConfig {
    id: u32,
    src_amount: u64,
    min_dst_amount: u64,
    estimated_dst_amount: u64,
    expiration_time: u32,
    src_asset_is_native: bool,
    dst_asset_is_native: bool,
    fee: FeeConfig,
    dutch_auction_data: AuctionData,
    cancellation_auction_duration: u32,
```

```
}
```

Recommendations:

Consider tracking the filled amount in the `escrow` PDA.

1inch: We do not create an escrow PDA to store the remaining tokens and prefer to leave the current behavior as is, since it does not disrupt the order lifecycle. This way, the maker pays lower fees when calling create.

Zenith: Acknowledged by client.

[L-2] Varying CU due to search for canonical bump could cause `fill()` to exceed CU limit

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

Target

- programs/fusion-swap/src/lib.rs#L484
- programs/fusion-swap/src/lib.rs#L515

Description:

The `fill` instruction uses `bump` instead of a stored bump in the account constraints for `resolver_access` and `escrow`. This causes it to search for the canonical bumps for every execution.

If the bumps of the `resolver_access` and `escrow` are 'far away', it could require a high amount of CU to find the canonical bump as described here.

That could cause the tx to exceed the standard CU limit set as `fill` will use more CU than creation of the PDAs. When that happens, it will require re-executing the tx with a higher CU limit.

```
pub struct Fill<'info> {
    /// `taker`, who buys `src_mint` for `dst_mint`
    #[account(mut, signer)]
    taker: Signer<'info>,
    /// Account allowed to fill the order
    #[account(
        seeds = [whitelist::RESOLVER_ACCESS_SEED, taker.key().as_ref()],
        bump,
        seeds::program = whitelist::ID,
    )]
    resolver_access: Account<'info, whitelist::ResolverAccess>,

    ...

    /// Account to store order conditions
    #[account(
        seeds = [
```

```
            "escrow".as_bytes(),
            maker.key().as_ref(),
            &order_hash(
                &order,
                protocol_dst_acc.clone().map(|acc| acc.key()),
                integrator_dst_acc.clone().map(|acc| acc.key()),
                src_mint.key(),
                dst_mint.key(),
                maker_receiver.key(),
            )?,
        ],
        bump,
    )]
    /// CHECK: check is not needed here as we never initialize the account
    escrow: UncheckedAccount<'info>,
```

Recommendations:

During PDA creation, store the canonical bumps in the account, and then load it for the `fill` instruction.

```
pub struct Fill<'info> {
    /// `taker`, who buys `src_mint` for `dst_mint`
    #[account(mut, signer)]
    taker: Signer<'info>,
    /// Account allowed to fill the order
    #[account(
        seeds = [whitelist::RESOLVER_ACCESS_SEED, taker.key().as_ref()],
-        bump,
+       resolver_access.bump,
        seeds::program = whitelist::ID,
    )]
    resolver_access: Account<'info, whitelist::ResolverAccess>,

   ...

    /// Account to store order conditions
    #[account(
        seeds = [
            "escrow".as_bytes(),
            maker.key().as_ref(),
            &order_hash(
                &order,
                protocol_dst_acc.clone().map(|acc| acc.key()),
                integrator_dst_acc.clone().map(|acc| acc.key()),
```

```
                src_mint.key(),
                dst_mint.key(),
                maker_receiver.key(),
            )?,
        ],
-       bump,
+       escrow.bump,
    )]
    /// CHECK: check is not needed here as we never initialize the account
    escrow: UncheckedAccount<'info>,
```

1inch: Storing the canonical bump in the resolver_access account is implemented in PR #74. We prefer not to create an additional escrow account and will leave current behavior as is.

Zenith: Verified.

[L-3] Missing validation of the `AuctionData`

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

Target

- programs/fusion-swap/src/auction.rs#L9-L15

Description:

The `AuctionData` stores the data for the dutch auction, which is used on the orders.

```
pub struct AuctionData {
    pub start_time: u32,
    pub duration: u32,
    pub initial_rate_bump: u16,
    pub points_and_time_deltas: Vec<PointAndTimeDelta>,
}
```

The `start_time` specifies when an auction will start. However, it is not verified that this timestamp is actually in the future.

The `duration` specifies the duration of the auction. It is never verified that this is greater than 0 on creation.

The `initial_rate_bump` specifies the start bump of the auction.It is never verified that this is greater than 0.

The `points_and_time_deltas` tracks the gradual decrease of the bumps. According to the documentation, it should contain 6 steps in descending order. However, this is not verified when the order is created.

Recommendations:

We recommend adding additional checks in `create` to ensure the AuctionData is valid.

1Inch: The backend will validate auction parameters. Orders with malicious auction data will not be tracked or shared with resolvers.

Zenith: Acknowledged due to check being done off-chain.

[L-4] Fees are rounded down

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

Target

- programs/fusion-swap/src/lib.rs#L761-L776

Description:

In the calculation of the fees in `get_fee_amounts` all fees are rounded down, resulting in minor losses to the protocol on each trade.

```
fn get_fee_amounts(
    integrator_fee: u16,
    protocol_fee: u16,
    surplus_percentage: u8,
    dst_amount: u64,
    estimated_dst_amount: u64,
) → Result<(u64, u64, u64)> {
    let integrator_fee_amount = dst_amount
        .mul_div_floor(integrator_fee as u64, BASE_1E5)
        .ok_or(ProgramError::ArithmeticOverflow)?;

    let mut protocol_fee_amount = dst_amount
        .mul_div_floor(protocol_fee as u64, BASE_1E5)
        .ok_or(ProgramError::ArithmeticOverflow)?;

    let actual_dst_amount = (dst_amount - protocol_fee_amount)
        .checked_sub(integrator_fee_amount)
        .ok_or(ProgramError::ArithmeticOverflow)?;

    if actual_dst_amount > estimated_dst_amount {
        protocol_fee_amount += (actual_dst_amount - estimated_dst_amount)
            .mul_div_floor(surplus_percentage as u64, BASE_1E2)
            .ok_or(ProgramError::ArithmeticOverflow)?;
    }

    Ok((
        protocol_fee_amount,
```

```
        integrator_fee_amount,
        dst_amount - integrator_fee_amount - protocol_fee_amount,
    ))
}
```

Recommendations:

We recommend ceiling the division that calculates the fees.

1inch: Acknowledged

[L-5] Complete `fill()` can be DoS by spamming dust fill

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

Target

- programs/fusion-swap/src/lib.rs#L134

Description:

Takers can choose to partially or completely fill the order by setting fill `amount` to `escrow_src_ata.amount`. When an complete fill occurs, the `escrow_src_ata` is closed on behalf of the maker.

A complete fill is likely to occur when the order is small or becomes small enough, which means that the `escrow_src_ata` will most probably be closed without requiring the maker to cancel it.

However, a rogue taker can prevent complete fills from occurring by spamming dust fill to reduce the `escrow_src_ata.amount`. This will cause complete fill to fail as the requested fill `amount` will be greater than what is available.

The impact of this issue is that the maker will not be able to get the entire order filled and has to cancel the order to close the `escrow_src_ata`.

The attack cost is inexpensive in Solana as the attacker just need to call `fill()` for every slot over the auction duration.However, it is noted that the taker is whitelisted and trusted, hence the low likelihood.

```
pub fn fill(ctx: Context<Fill>, order: OrderConfig, amount: u64) →
    Result<()> {
    require!(
        Clock::get()?.unix_timestamp ⩽ order.expiration_time as i64,
        FusionError::OrderExpired
    );

    require!(
        amount ⩽ ctx.accounts.escrow_src_ata.amount,
        FusionError::NotEnoughTokensInEscrow
    );
```

```
    require!(amount != 0, FusionError::InvalidAmount);
```

Recommendations:

Consider adjusting the fill amount to what is available when `amount > escrow_src_ata.amount`.

1inch: Takers are trusted entities who have passed a KYC process. Any taker who attempts to cause a DoS by dust fills will be banned and could face legal consequences.

Zenith: Acknowledged by client.

[L-6] Fees are not verified to be less than 100%

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

Target

- programs/fusion-swap/src/lib.rs#L44

Description:

The `protocol_fee` and the `integrator_fee` are denominated in 1e5. '

```
let mut protocol_fee_amount = dst_amount
    .mul_div_floor(protocol_fee as u64, BASE_1E5)
    .ok_or(ProgramError::ArithmeticOverflow)?;

let actual_dst_amount: u64 = (dst_amount - protocol_fee_amount)
    .checked_sub(integrator_fee_amount)
    .ok_or(ProgramError::ArithmeticOverflow)?;
```

If the sum of both fees were more significant than 100%, all trades would fail.

Recommendations:

We recommend verifying that both `protocol_fee + integrator_fee ⩽ BASE_1E5`.

1inch: These checks will be performed by the backend. Orders with incorrect fees will not be recorded in the database or shared with resolvers. Implementing these checks on-chain would increase CU usage, and we prefer to avoid that.

Zenith: Acknowledged due to checks being done off-chain.

[L-7] Takers can fill expired orders

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

Target

- programs/fusion-swap/src/lib.rs#L123-L127

Description:

In `cancel_by_resolver()`, the resolver can only cancel the order when it has expired i.e. `current_timestamp ⩾ order.expiration_time`.

However, `fill()` allows taker to fill the order when `timestamp ⩽ order.expiration_time`. It should instead be `timestamp < order.expiration_time`.

```
pub fn fill(ctx: Context<Fill>, order: OrderConfig, amount: u64) →
   Result<()> {
    require!(
        Clock::get()?.unix_timestamp ⩽ order.expiration_time as i64,
        FusionError::OrderExpired
    );

...
```

```
pub fn cancel_by_resolver(
    ctx: Context<CancelByResolver>,
    order: OrderConfig,
    reward_limit: u64,
) → Result<()> {
    require!(
        order.fee.max_cancellation_premium > 0,
        FusionError::CancelOrderByResolverIsForbidden
    );
    let current_timestamp = Clock::get()?.unix_timestamp;
    require!(
        current_timestamp ⩾ order.expiration_time as i64,
        FusionError::OrderNotExpired
    );
```

Recommendations:

```
pub fn fill(ctx: Context<Fill>, order: OrderConfig, amount: u64) ->
Result<()> {
    require!(
        Clock::get()?.unix_timestamp ⩽ order.expiration_time as i64,
        Clock::get()?.unix_timestamp < order.expiration_time as i64,
        FusionError::OrderExpired
    );

...
```

1inch: This issue is resolved in PR #76, which also changes the comparison in the create function from ⩽ to <.

Zenith: Verified.

[L-8] Protocol allows Token2022 extension but does not handle or reject them

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

Target

- programs/fusion-swap/src/lib.rs#L437
- programs/fusion-swap/src/lib.rs#L536-L537

Description:

The `create` and `fill` instructions uses `TokenInterface` account type for the token program and this allows use of tokens that have token2022 extensions like TransferFee, TransferHooks, etc.

However, the program does not handle or reject such tokens. This could have undesirable consequences such as wrong calculation of the transfer amounts.

Recommendations:

This can be resolved by checking the token extensions and only allow supported extensions such as Metadata. A reference on such check can be found here.

Otherwise, fix the token program to the original Token Program if there are no plans to support Token 2022 extensions.

1inch: We will blacklist unsupported tokens and extensions on the backend. Only MetadataPointer and TokenMetadata will be permitted.

Zenith: To be resolved off-chain.