



Smart Contract Security Audit Report

1inch

1. Contents

1.	Contents	2
2.	General Information	3
2.1.	Introduction	3
2.2.	Scope of Work	3
2.3.	Threat Model.....	3
2.4.	Weakness Scoring	4
2.5.	Disclaimer	4
3.	Summary.....	5
3.1.	Suggestions	5
4.	General Recommendations	6
4.1.	Security Process Improvement	6
5.	Findings.....	7
5.1.	Incorrect escrow seed for partial fills.....	7
5.2.	DoS via escrow_ata pre-initialization	10
5.3.	Rescue start could be earlier than cancellation start.....	12
5.4.	The maker may front-run order fulfillment.....	12
5.5.	Safety deposit is too low	13
5.6.	Token-2022 with extensions could behave unexpectedly	13
6.	Appendix.....	15
6.1.	About us	15

2. General Information

This report contains information about the results of the security audit of the [1inch](#) (hereafter referred to as “Customer”) smart contracts, conducted by [Decurity](#) in the period from 13/06/2025 to 20/06/2025.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: [solana-crosschain-protocol](#) (commit [06ef253](#)). Re-testing was done for the commit [9572454](#).

The following contracts have been tested:

- `programs/**/src/*.rs`
- `common/src/*.rs`

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- Taker
- Maker

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered multiple high issues.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of October 10, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Incorrect escrow seed for partial fills	/programs/cross-chain-escrow-src/src/lib.rs	High	Fixed
DoS via escrow_ata pre-initialization	/programs/cross-chain-escrow-src/src/lib.rs, /programs/cross-chain-escrow-dst/src/lib.rs	High	Fixed
Rescue start could be earlier than cancellation start	/programs/cross-chain-escrow-src/src/lib.rs	Medium	Fixed
The maker may front-run order fulfillment	/programs/cross-chain-escrow-src/src/lib.rs	Low	Fixed
Safety deposit is too low	/common/src/escrow.rs	Low	Acknowledged
Token-2022 with extensions could behave unexpectedly	/common/src/escrow.rs	Info	Acknowledged

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Incorrect escrow seed for partial fills

Risk Level: High

Status: Fixed in the commit [6e0ba09](#)

Contracts:

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

Description:

When taker initiates escrow the following account is initialized:

```
#[account(  
  init,  
  payer = taker,  
  space = constants::DISCRIMINATOR_BYTES + EscrowSrc::INIT_SPACE,  
  seeds = [  
    "escrow".as_bytes(),  
    order.order_hash.as_ref(),  
    &get_escrow_hashlock(  
      order.hashlock,  
      merkle_proof.clone()  
    ),  
    order.creator.as_ref(),  
    taker.key().as_ref(),  
    mint.key().as_ref(),  
    amount.to_be_bytes().as_ref(),  
    order.safety_deposit.to_be_bytes().as_ref(),  
    order.rescue_start.to_be_bytes().as_ref(),  
  ],  
  bump,  
)]  
escrow: Box<Account<'info, EscrowSrc>>,
```

Where amount that is used as seed is the amount that is filled by taker. After that during function execution escrow account is assigned the following values:

```
ctx.accounts.escrow.set_inner(EscrowSrc {  
  order_hash: order.order_hash,  
  hashlock,  
  maker: order.creator,  
  taker: ctx.accounts.taker.key(),  
  token: order.token,
```

```
        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,
        dst_amount: get_dst_amount(order.dst_amount, &dutch_auction_data)?,
    });
```

where `amount` is `order.amount` not the amount that is provided by taker and used as seed.

After that when trying to interact with the escrow the following escrow seeds are used during escrow account validation:

```
#[account(
    mut,
    seeds = [
        "escrow".as_bytes(),
        escrow.order_hash.as_ref(),
        escrow.hashlock.as_ref(),
        escrow.maker.as_ref(),
        escrow.taker.as_ref(),
        mint.key().as_ref(),
        escrow.amount.to_be_bytes().as_ref(),
        escrow.safety_deposit.to_be_bytes().as_ref(),
        escrow.rescue_start.to_be_bytes().as_ref(),
    ],
    bump,
)]
escrow: Box<Account<'info, EscrowSrc>>>,
```

However, in case fill was partial `escrow.amount` will mismatch with the actual seed that was used during initialization, because it was overridden with the `order.amount`.

Such behaviour results in inability to provide a valid escrow account and withdraw funds, cancel, etc. Only rescue can be performed.

Here is an example test that should be placed in the following file: `solana-crosschain-protocol/programs/cross-chain-escrow-src/tests/test_cross_chain_escrow_src.rs`

```
mod test_partial_fill_escrow_creation {
    // ...
    #[test_context(TestState)]
    #[tokio::test]
    async fn test_create_two_escrows_for_full_order_withdraw_audit(test_state:
    &mut TestState) {
```

```
        let (order, order_ata) =
            create_order_for_partial_fill(test_state).await;
            prepare_resolvers(test_state,
                &[test_state.recipient_wallet.keypair.pubkey()]).await;
            let escrow_amount = DEFAULT_ESCROW_AMOUNT /
                DEFAULT_PARTS_AMOUNT_FOR_MULTIPLE;
            let (escrow1, escrow_ata1) =
                create_escrow_for_partial_fill(test_state, escrow_amount).await;
            let (escrow2, escrow_ata2) = create_escrow_for_partial_fill(
                test_state,
                DEFAULT_ESCROW_AMOUNT - escrow_amount, // full fill
            )
                .await;

            set_time(
                &mut test_state.context,
                test_state.init_timestamp + DEFAULT_PERIOD_DURATION *
                PeriodType::Withdrawal as u32,
            );

            let rent_recipient = test_state.recipient_wallet.keypair.pubkey();
            local_helpers::test_escrow_withdraw_audit(test_state, escrow1,
                escrow_ata1, rent_recipient).await
        }
        // ...
    }

mod local_helpers {
    // ...
    pub async fn test_escrow_withdraw_audit<T: EscrowVariant<S> + 'static, S:
        TokenVariant>(test_state: &mut TestStateBase<T, S>, escrow: Pubkey,
            escrow_ata: Pubkey, rent_recipient: Pubkey) {
        let transaction = T::get_withdraw_tx(test_state, &escrow,
            &escrow_ata);

        test_state.client.process_transaction(transaction).await.expect_success(); //
        @audit reverts with "seeds constraint was violated"
    }
    // ...
}
```

Test output:

```
running 1 test
test test_partial_fill_escrow_creation::test_create_two_escrows_for_full_order_withdraw_audit ... FAILED

failures:

---- test_partial_fill_escrow_creation::test_create_two_escrows_for_full_order_withdraw_audit stdout ----
Instruction: Create
Instruction: Initialize
Instruction: Register
Instruction: CreateEscrow
Instruction: CreateEscrow
Instruction: Withdraw
AnchorError caused by account: escrow. Error Code: ConstraintSeeds. Error Number: 2006. Error Message: A seeds constraint was violated.
Left:
7qM83UJM95dZbkjx8pxipXQ7GUNMHfjd3JUmCvSVti6f
Right:
818w1SgNmTjqQEu3yWTTxqg198kC4T2zX5vsdsF94HDu

thread 'test_partial_fill_escrow_creation::test_create_two_escrows_for_full_order_withdraw_audit' panicked at common/tests/src/helpers.rs:764:14:
called 'Result::unwrap()' on an 'Err' value: TransactionError(InstructionError(0, Custom(2006)))
note: run with 'RUST_BACKTRACE=1' environment variable to display a backtrace

failures:
  test_partial_fill_escrow_creation::test_create_two_escrows_for_full_order_withdraw_audit
test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 measured; 192 filtered out; finished in 0.25s

error: test failed, to rerun pass '-p cross-chain-escrow-src --test test_cross_chain_escrow_src'
```

Remediation:

Consider not overriding escrow amount with `order.amount`.

5.2. DoS via escrow_ata pre-initialization

Risk Level: High

Status: Fixed in the commit [86e2c28](#)

Contracts:

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

Location: Function: `create_escrow()`, `create()`.

Description:

In the `src escrow` program, the `create_escrow()` function's account constraints include the escrow and the corresponding `escrow_ata` accounts, which must not exist prior to the function call and are intended to be initialized during the instruction execution via the `init` constraint. However, the `escrow_ata` account can be pre-initialized before the escrow account using a direct call to the Associated Token Account Program. This makes it impossible to invoke the `create_escrow()` function if the `escrow_ata` account has already been initialized.

This function is called by a resolver to begin order fulfillment when the auction price is deemed reasonable. A malicious resolver could exploit this by pre-initializing the `escrow_ata` accounts for all potential resolvers and hashlocks, effectively blocking them from calling `create_escrow()`. This allows

the malicious resolver to delay execution until the price reaches its minimum, enabling them to complete the order with a significant arbitrage advantage.

Test:

```
mod token_module {
  // ...
  mod test_order_creation {
    // ...
    #[test_context(TestState)]
    #[tokio::test]
    async fn test_escrow_audit_token_dos(
      test_state: &mut TestState,
    ) {
      prepare_resolvers(test_state,
        &[test_state.recipient_wallet.keypair.pubkey()])
        .await;
      local_helpers::test_escrow_audit_token_dos(test_state).await;
    }
    // ...
  }
}

mod local_helpers {
  // ...
  pub async fn test_escrow_audit_token_dos<S: TokenVariant>(
    test_state: &mut TestStateBase<SrcProgram, S>,
  ) {
    create_order(test_state).await;
    let (escrow, escrow_ata, tx) = create_escrow_data(test_state);

    S::initialize_spl_associated_account(&mut test_state.context,
      &test_state.token, &escrow).await;

    test_state
      .client
      .process_transaction(tx)
      .await.expect_success(); // @audit reverts
  }
  // ...
}
```

Remediation:

Use `init_if_needed` instead of `init` in ATA accounts validation.

5.3. Rescue start could be earlier than cancellation start

Risk Level: Medium

Status: Fixed in the commit [2e4dd99](#)

Contracts:

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

Location: Function: create_escrow.

Description:

In case rescue_start is placed before public_cancellation_start, the taker may take all of the makers tokens without a need to actually execute an order.

Remediation:

Consider checking that rescue_start is placed behind public_cancellation_start.

5.4. The maker may front-run order fulfillment

Risk Level: Low

Status: Fixed in the commit [9572454](#)

Contracts:

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

Description:

The maker may execute cancel_order at any time—for example, to front-run order fulfillment, cancel their order, and recreate it using the same seeds but with different timestamps. This may result in the taker fulfilling an order with timestamps different from what they expected. If these timestamps are not further validated, it could lead to the creation of an order with incorrect timestamps on the destination chain. For instance, the src_cancellation_timestamp could end up being greater than cancellation_start, because it was extracted from the wrong order.

Remediation:

Consider either verifying all information on the resolver side after escrow creation, checking the order_hash on the contract side (which should include all relevant data), or including all provided arguments in the order seeds.

5.5. Safety deposit is too low

Risk Level: Low

Status: Client's comment: We are currently comfortable proceeding with a limited safety deposit and do not plan to change the current behavior.

Contracts:

- /common/src/escrow.rs

Location: Function: create().

Description:

Unlike the implementation on the EVM, on Solana, a safety deposit is limited by the cost of the escrow PDA rent. Accordingly, when creating an escrow, the resolver does not allocate a separate amount for the deposit, spending only a fee on the PDA rent, which can later be reclaimed. Since the cost of renting an account on Solana is low, this significantly reduces the incentive for resolvers to perform further actions within the required time intervals and increases the likelihood of expired and unfulfilled positions.

Remediation:

Require the resolver to deposit a separate safety amount that can be redeemed later if the required actions are executed on time.

5.6. Token-2022 with extensions could behave unexpectedly

Risk Level: Info

Status: The backend should filter out orders with tokens supporting extensions that can block token transfers from escrow.

Contracts:

- /common/src/escrow.rs

Description:

The protocol supports both SPL Token and Token-2022 standards. However, the account closing logic in escrow.rs does not account for the stricter requirements of Token-2022 when certain extensions are enabled. This can cause the close_account instruction to fail, leading to a denial-of-service for core functions like withdraw(), cancel(), and rescue_funds().

The reason is the `close_account` instruction has different prerequisites for SPL Token and Token-2022.

- **SPL Token:** A non-native token account can be closed by its authority if its token balance is zero (`account.amount == 0`).
- **Token-2022:** In addition to a zero token balance, several other conditions may apply depending on the enabled extensions:
- **CPI Guard:** When closing via CPI, the `lamport` destination must be the token account's owner.
- **Confidential Transfer:** The pending and available balances must both be zero.
- **Confidential Transfer Fee / Transfer Fee:** The withheld fee amount must be zero.

Remediation:

Token-2022 extension provides `closable` methods to check condition to avoid raising the error. Consider implementing additional logic to handle tokens with extensions correctly.

Reference: <https://blog.offside.io/i/148944074/closing-the-token-account>

6. Appendix

6.1. About us

The [Decurity](#) team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.