SHERLOCK

Security Review For 1inch

Collaborative Audit Prepared For:	1inch
Lead Security Expert(s):	0xeix
	bin2chen
Date Audited:	July 21 - July 24, 2025
Final Commit:	4b9053e

Introduction

TBA

Scope

Repository: 1inch/solana-crosschain-protocol

Audited Commit: f9268b669a666bd73520cd790bfe6741363e038b

Final Commit: 4b9053e069e4203b7bb75d095f51379024963244

Files:

- common/src/constants.rs
- common/src/error.rs
- common/src/escrow.rs
- common/src/lib.rs
- common/src/timelocks.rs
- common/src/utils.rs
- programs/cross-chain-escrow-dst/src/lib.rs
- programs/cross-chain-escrow-src/src/auction.rs
- programs/cross-chain-escrow-src/src/lib.rs
- programs/cross-chain-escrow-src/src/merkle_tree.rs
- programs/whitelist/src/error.rs
- programs/whitelist/src/lib.rs

Final Commit Hash

4b9053e069e4203b7bb75d095f51379024963244

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
1	1	5

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: missing check creator when executing rescue_funds()

Source: https://github.com/sherlock-audit/2025-07-1inch-july-21st/issues/14

Summary

`rescue_funds()` should now only be executable by `creator`, but currently there is no check to verify that the executor is the actual creator.

Vulnerability Detail

The new version removed `creator` from EscrowDst.seeds, and there is no check within the method to verify that the method executor is `escrow.creator`.

```
pub struct RescueFunds<'info> {
    #[account(
        mut, // Needed because this account receives lamports from closed token
         ↪  account.
    )]
    creator: Signer<'info>,
    /// CHECK: This account is used to check its pubkey to match the one stored in
     ↪  the escrow account seeds
    recipient: AccountInfo<'info>,
    mint: Box<InterfaceAccount<'info, Mint>>,
    /// CHECK: We don't accept escrow as 'Account<'info, Escrow>' because it may be
     ↪  already closed at the time of rescue funds.
    #[account(
        seeds = [
            "escrow".as_bytes(),
            order_hash.as_ref(),
            hashlock.as_ref(),
            recipient.key().as_ref(),
            escrow_amount.to_be_bytes().as_ref(),
        ],
        bump,
    )]
    escrow: Box<Account<'info, EscrowDst>>,
```

```
pub fn rescue_funds(
    ctx: Context<RescueFunds>,
    order_hash: [u8; 32],
    hashlock: [u8; 32],
    escrow_amount: u64,
    rescue_amount: u64,
) -> Result<()> {
    let recipient_pubkey = ctx.accounts.recipient.key();

    let rescue_start = if !ctx.accounts.escrow.data_is_empty() {
        let escrow_data =
            EscrowDst::try_deserialize(&mut
             ↪   &ctx.accounts.escrow.data.borrow()[..])?;
        Some(Timelocks(U256(escrow_data.timelocks)).rescue_start(constants::RESCUE_
         ↪   DELAY)?)
    } else {
        None
    };
```

This way when `RESCUE_DELAY` is exceeded, anyone can impersonate the creator to steal the token.

Impact

anyone can impersonate the creator to steal the token.

Code Snippet

https://github.com/sherlock-audit/2025-07-1inch-july-21st/blob/a55f4e48ff6c9cb5d4a9d1e3c2a1a28682913a37/solana-crosschain-protocol/programs/cross-chain-escrow-dst/src/lib.rs#L204-L207

Recommendation

check `accounts.creator == escrow.creator`

```
    pub fn rescue_funds(
        ctx: Context<RescueFunds>,
        order_hash: [u8; 32],
        hashlock: [u8; 32],
        escrow_amount: u64,
        rescue_amount: u64,
    ) -> Result<()> {
        let recipient_pubkey = ctx.accounts.recipient.key();

        let rescue_start = if !ctx.accounts.escrow.data_is_empty() {
            let escrow_data =
                EscrowDst::try_deserialize(&mut
                 ↪   &ctx.accounts.escrow.data.borrow()[..])?;
+           require!(
+               ctx.accounts.creator.key() == escrow_data.creator.key(),
+               EscrowError::InvalidRescueStart
+           );
```

```
            Some(Timelocks(U256(escrow_data.timelocks)).rescue_start(constants::RES⌟
             ↪  CUE_DELAY)?)
        } else {
            None
        };
```

Discussion

SevenSwen

Fixed in https://github.com/1inch/solana-crosschain-protocol/pull/115/commits/40c73959a603140d4a465b739847325b5705a8e6

bin2chen66

Fixed modified `escrow.seeds =... + creator_pubkey + ...` ensures that only the creator can call it.

Issue M-1: cancel_escrow() missing check mint

Source: https://github.com/sherlock-audit/2025-07-1inch-july-21st/issues/16

Summary

a malicious taker can specify the wrong mint in `cancel_escrow()`, then and steal the real escrow.token via `rescue_funds_for_escrow()`

Vulnerability Detail

`cancel_escrow()` does not check `mint`, so the taker can specify any mint

```
pub struct CancelEscrow<'info> {
    #[account(
        mut, // Needed because this account receives lamports from rent
        constraint = taker.key() == escrow.taker @ EscrowError::InvalidAccount
    )]
    taker: Signer<'info>,
    /// CHECK: this account is used only to receive lamports and to check its
     ↪  pubkey to match the one stored in the escrow account
    #[account(
        mut, // Needed because this account receives lamports if the token is native
        constraint = maker.key() == escrow.maker @ EscrowError::InvalidAccount
    )]
    maker: AccountInfo<'info>,
@>  mint: Box<InterfaceAccount<'info, Mint>>,
    #[account(
        mut,
        close = taker,
        seeds = [
            "escrow".as_bytes(),
            escrow.order_hash.as_ref(),
            escrow.hashlock.as_ref(),
            taker.key().as_ref(),
            escrow.amount.to_be_bytes().as_ref(),
        ],
        bump = escrow.bump,
    )]
    escrow: Box<Account<'info, EscrowSrc>>,
```

`cancel_escrow()` was originally intended to return `escrow.token` to `maker`.

At that time, `taker` could maliciously specify a worthless and incorrect `mint`.

After `cancel_escrow()` execution, `escrow` is closed and `taker` can bypass the `RESCUE_DELAY` restriction and steal the real `escrow.token` by `rescue_funds_for_escrow()`.

Note: Withdraw / PublicWithdraw / PublicCancelEscrow Similar issue

Impact

`escrow.token` can be maliciously transferred away.

Code Snippet

https://github.com/sherlock-audit/2025-07-1inch-july-21st/blob/a55f4e48ff6c9cb5d4a9d1e3c2a1a28682913a37/solana-crosschain-protocol/programs/cross-chain-escrow-src/src/lib.rs#L855

Recommendation

```
+   #[account(
+       constraint = mint.key() == escrow.token @ EscrowError::InvalidMint
+   )]
    mint: Box<InterfaceAccount<'info, Mint>>,
    #[account(
        mut,
        close = taker,
        seeds = [
            "escrow".as_bytes(),
            escrow.order_hash.as_ref(),
            escrow.hashlock.as_ref(),
            taker.key().as_ref(),
            escrow.amount.to_be_bytes().as_ref(),
        ],
        bump = escrow.bump,
    )]
    escrow: Box<Account<'info, EscrowSrc>>,
```

Discussion

SevenSwen

Fixed in https://github.com/1inch/solana-crosschain-protocol/pull/115/commits/40c73959a603140d4a465b739847325b5705a8e6

bin2chen66

Fixed `Withdraw / PublicWithdraw / CancelEscrow / PublicCancelEscrow` add constraint `mint.key() == escrow.token`

Issue L-1: EscrowDst.seeds is vulnerable to DoS attacks

Source: https://github.com/sherlock-audit/2025-07-1inch-july-21st/issues/13

Summary

The new version removed `creator` from EscrowDst.seeds, making this seed vulnerable to DoS attacks.

Vulnerability Detail

The new version of `EscrowDst.seeds = order_hash + hashlock + recipient + amount` removed `creator` and `mint`.

```
pub struct RescueFunds<'info> {
    #[account(
        mut, // Needed because this account receives lamports from closed token
         ↪  account.
    )]
    creator: Signer<'info>,
    /// CHECK: This account is used to check its pubkey to match the one stored in
    ↪  the escrow account seeds
    recipient: AccountInfo<'info>,
    mint: Box<InterfaceAccount<'info, Mint>>,
    /// CHECK: We don't accept escrow as 'Account<'info, Escrow>' because it may be
    ↪  already closed at the time of rescue funds.
    #[account(
        seeds = [
            "escrow".as_bytes(),
            order_hash.as_ref(),
            hashlock.as_ref(),
            recipient.key().as_ref(),
            escrow_amount.to_be_bytes().as_ref(),
        ],
        bump,
    )]
```

This allows malicious users to easily use the same `order_hash + hashlock + recipient + amount` to cause `seeds` conflicts and carry out DoS attacks (using worthless `mint`).

Impact

Dos attacks `cross_chain_escrow_dst.creator()`

Code Snippet

https://github.com/sherlock-audit/2025-07-1inch-july-21st/blob/a55f4e48ff6c9cb5d4a9d1e3c2a1a28682913a37/solana-crosschain-protocol/programs/cross-chain-escrow-dst/src/lib.rs#L440

Recommendation

keep seeds = `order_hash + hashlock + creator + mint + recipient + amount`.

Discussion

SevenSwen

Fixed in https://github.com/1inch/solana-crosschain-protocol/commit/40c73959a603140d4a465b739847325b5705a8e6

We added the missing checks on mint as a constraint, but removed it, as well as the recipient and safety_deposit parameters from the seeds as redundant.

Our goal was for the parameters in the seeds to ensure the uniqueness of the escrow so that a user or resolver could create necessary escrows with similar parameters (for example, if a user wants to swap the same tokens twice for the same amount).

However, from a security standpoint, we prefer not to add these parameters to the seeds in place of constraints to avoid complicating seed formation for the backend and to keep parameter checks in the contract more readable.

bin2chen66

Fixed Modify `escrow.seeds= ... + creator.key() + ...` avoids not being able to maliciously DOS other users

Issue L-2: cross_chain_escrow_dst instruction missing check mint

Source: https://github.com/sherlock-audit/2025-07-1inch-july-21st/issues/15

Summary

`Withdraw/PublicWithdraw/Cancel` instruction missing check for `mint == escrow.token`

Vulnerability Detail

Taking `PublicWithdraw` as an example, there is currently no check for `mint`, which allows `payer` to arbitrarily specify `mint`.

```
#[derive(Accounts)]
pub struct PublicWithdraw<'info> {
...
    #[account(mut)]
    payer: Signer<'info>,
    #[account(
        seeds = [whitelist::RESOLVER_ACCESS_SEED, payer.key().as_ref()],
        bump = resolver_access.bump,
        seeds::program = whitelist::ID,
    )]
    resolver_access: Account<'info, whitelist::ResolverAccess>,
    mint: Box<InterfaceAccount<'info, Mint>>,
    #[account(
        mut,
        close = creator,
        seeds = [
            "escrow".as_bytes(),
            escrow.order_hash.as_ref(),
            escrow.hashlock.as_ref(),
            escrow.recipient.key().as_ref(),
            escrow.amount.to_be_bytes().as_ref(),
        ],
        bump = escrow.bump,
    )]
    escrow: Box<Account<'info, EscrowDst>>,
```

Note: `Withdraw/PublicWithdraw/Cancel` similar issue

Impact

Malicious specification of `mint` prevents `PublicWithdraw` from working properly.

Code Snippet

https://github.com/sherlock-audit/2025-07-1inch-july-21st/blob/a55f4e48ff6c9cb5d4a9d1e3c2a1a28682913a37/solana-crosschain-protocol/programs/cross-chain-escrow-dst/src/lib.rs#L353

Recommendation

add `constraint = mint.key() == escrow.token`

```
+   #[account(
+       constraint = mint.key() == escrow.token @ EscrowError::InvalidMint
+   )]
    mint: Box<InterfaceAccount<'info, Mint>>,
```

Discussion

SevenSwen

Fixed in https://github.com/1inch/solana-crosschain-protocol/commit/40c73959a603140d4a465b739847325b5705a8e6

bin2chen66

Fixed `Withdraw/PublicWithdraw/Cancel` add constraint `mint.key() == escrow.token`

Issue L-3: Excessive checking leads to more computationally expensive execution

Source: https://github.com/sherlock-audit/2025-07-1inch-july-21st/issues/18

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

At the moment, there are several unnecessary checks present for the ATAs across the protocol.

Vulnerability Detail

Excessive checking leads to more computationally expensive execution. The token instruction will itself verify:

- mint is the same for creator_ata and escrow_ata
- the signer of the tx has to be the creator, so here the check authority = creator is not needed as tx will not pass if somebody will try to take the funds from the different ATA

Impact

Constraining accounts more than it needs leads to an increased tx cost of the execution.

Code Snippet

```
#[account(
     mut,
     associated_token::mint = mint,
     associated_token::authority = creator,
     associated_token::token_program = token_program
 )]
 /// Account to store creator's tokens (Optional if the token is native)
 creator_ata: Option<Box<InterfaceAccount<'info, TokenAccount>>>,
```

Tool Used

Manual Review

Recommendation

Overall, the suggested fix is the following: just basically remove the constraints for the creator_ata:

```
/// Account to store creator's tokens (Optional if the token is native)
creator_ata: Option<Box<InterfaceAccount<'info, TokenAccount>>>,
```

Discussion

SevenSwen

If we remove these checks as you suggest, two problems will arise:

1) There will be no consistency in the protocol's behavior between the create and cancel functions. In that case, we would also have to remove these checks from cancel, but then it seems a security vulnerability could appear, since it would be possible to pass an account different from the one the user provided during create. To ensure that the same account is used, we would have to store it in the order account, which means an extra 32 bytes and additional lamports for rent.
2) This will hurt readability. Someone seeing the code for the first time would have to guess what restrictions exist for this account based on indirect signs. It seems better to specify them explicitly.

rodiontr

> If we remove these checks as you suggest, two problems will arise:
> 1. There will be no consistency in the protocol's behavior between the create and cancel functions. In that case, we would also have to remove these checks from cancel, but then it seems a security vulnerability could appear, since it would be possible to pass an account different from the one the user provided during create. To ensure that the same account is used, we would have to store it in the order account, which means an extra 32 bytes and additional lamports for rent.
> 2. This will hurt readability. Someone seeing the code for the first time would have to guess what restrictions exist for this account based on indirect signs. It seems better to specify them explicitly.

I don't see the problem of removing it just from the create instruction - I mean there is nothing wrong of removing it from only one instruction if there are security implications in another when removing like that. And there is no any inconsistency when removing just from the `create` because there is no necessity to remove from the `cancel` just because the functions are mirrored

SevenSwen

Your suggestion can be considered as a minor CU optimization, and in that case, indeed, there is no need to make changes to other functions. However, if we remove the specified checks, we are actually expanding the protocol's capabilities, since a resolver will now be able to pass not just an ATA but any token_account in create for dst_escrow. This can actually be useful, for example, in cases involving WSOL accounts. Since unwrapping WSOL requires closing the account, and our backend is currently forced to use the same ATA for all escrows of different users, it has to constantly recreate the native token ATA every time it needs to unwrap WSOL. If we remove the associated_token checks, it would be possible to create a separate native token account for each user and unwrap tokens whenever needed.

In other words, I see your proposal not just as a CU optimization but as a protocol feature enhancement. However, to make this truly convenient, we would need a closed system where the backend could use token accounts other than ATA in all cases, not only during escrow_dst creation. This, however, looks like a protocol upgrade that hasn't been implemented in the current version and will likely be deferred to future versions.

Coming back to the point that your suggestion could be treated merely as a CU optimization – we will discuss this once again, but for now, we are inclined to believe that having the explicit check is preferable for us compared to the CU savings in this particular case.

rodiontr

> he point that your suggestion could be treated merely as a CU optimization – we will discuss this once again, but for now, we are inclined to believe that having the explicit check is preferable for us compared to the CU savings in this particular case.

agree with you, this is basically an informational finding for the purpose of saving the CU units

Issue L-4: `safety_deposit` is not guaranteed to have a reasonable value allowing to bypass it

Source: https://github.com/sherlock-audit/2025-07-1inch-july-21st/issues/19

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

At the moment, the checks that are performed on the `safety_deposit` value are insufficient and it allows for the users to specify such a deposit that won't cover the transaction cost of the public execution.

Vulnerability Detail

The only checks for `safety_deposit` try to ensure that it's whether non-zero or it's `safety_deposit to be <= rent_exempt_reserve`.

Impact

Medium

Code Snippet

https://github.com/sherlock-audit/2025-07-1inch-july-21st/blob/main/solana-crosschain-protocol/programs/cross-chain-escrow-dst/src/lib.rs#L44-47

```
require!(
    amount != 0 && safety_deposit != 0,
    EscrowError::ZeroAmountOrDeposit
```

Tool Used

Users can specify `safety_deposit` to be slightly greater than 0 that will not even cover the tx fees for the public methods execution.

Recommendation

Introduce a hardcoded ”minimum value” to compare against or calculate it as a percentage of the remaining lamports.

Discussion

SevenSwen

Resolvers undergo a KYB procedure, and if they create an escrow with parameters different from those specified on the src chain (including the safety_deposit amount), they will be removed from the list of resolvers.

rodiontr

> Resolvers undergo a KYB procedure, and if they create an escrow with parameters different from those specified on the src chain (including the safety_deposit amount), they will be removed from the list of resolvers.

understood, i just didn't have this information, can we leave this as informational in this case as the mitigation described still suggests a good sanity check ?

SevenSwen

I didn't quite understand which sanity check you are suggesting. Which specific constant should we hardcode instead of zero for this check?

rodiontr

> I didn't quite understand which sanity check you are suggesting. Which specific constant should we hardcode instead of zero for this check?

the safety deposit - otherwise, what's the reason to check it at all? As in the code it's currently checked to be non-zero

SevenSwen

I agree that the current check doesn't bring much value and can be removed without any real downside to the protocol. However, I still think it's a bad idea to strengthen this check by replacing 0 with some non-zero value. At the very least, I don't know how to calculate an appropriate minimum safety deposit that could be hardcoded and still be meaningfully aligned with the transaction costs that the resolver would incur.

rodiontr

> I agree that the current check doesn't bring much value and can be removed without any real downside to the protocol. However, I still think it's a bad idea to strengthen this check by replacing 0 with some non-zero value. At the very least, I don't know how to calculate an appropriate minimum safety deposit that could be hardcoded and still be meaningfully aligned with the transaction costs that the resolver would incur.

agree with that, I think the possible solution could be just setting some reasonable value that's slightly bigger than the usual tx fee cost and add an `update_safe_deposit()` function to dynamically update if needed similar how there is a minimum deposit value in other protocols, for example

Issue L-5: `sync_native()` is not called on the wSOL transfer

Source: https://github.com/sherlock-audit/2025-07-1inch-july-21st/issues/20

Summary

At the moment, the sync on the transferred SOL is not performed to reflect the wSOL amount in the `escrow_ata` on the `escrow-dst`.

Vulnerability Detail

Currently, the lamports are only transferred from the `creator` to the `escrow_ata` but not synced. This creates a situation where SPL token balance (the `amount` field in the token account) remains unchanged because the SPL token program doesn't automatically convert the deposited SOL into wSOL tokens.

Impact

The SOL tokens not converted into wSOL mint and therefore wrapped SOL tokens can be used in the following instructions.

Code Snippet

```
// Check if token is native (WSOL) and is expected to be wrapped
       if asset_is_native {
           // Transfer native tokens from creator to escrow_ata and wrap
           uni_transfer(
               &UniTransferParams::NativeTransfer {
                   from: ctx.accounts.creator.to_account_info(),
                   to: ctx.accounts.escrow_ata.to_account_info(),
                   amount,
                   program: ctx.accounts.system_program.clone(),
               },
               None,
           )?;
```

As you can see here, the comment indicates that the tokens are supposed to be synced similarly to how it's done on the `escrow-src`:

```
// Check if token is native (WSOL) and is expected to be wrapped
      if asset_is_native {
          // Transfer native tokens from creator to escrow_ata and wrap
          uni_transfer(
            &UniTransferParams::NativeTransfer {
                from: ctx.accounts.creator.to_account_info(),
                to: ctx.accounts.order_ata.to_account_info(),
                amount,
                program: ctx.accounts.system_program.clone(),
            },
            None,
        )?;

        anchor_spl::token::sync_native(CpiContext::new(
            ctx.accounts.token_program.to_account_info(),
            anchor_spl::token::SyncNative {
                account: ctx.accounts.order_ata.to_account_info(),
            },
        ))?;
```

Tool Used

Manual Review

Recommendation

Add the `sync_native()` instruction in the `create()` function of the `escrow-dst`.

Discussion

SevenSwen

This behavior is intentional and was designed so that the currency received in the escrow is exactly what is returned on the dst chain. It's not entirely clear what you consider a vulnerability—could you please describe it in more detail?

rodiontr

> This behavior is intentional and was designed so that the currency received in the escrow is exactly what is returned on the dst chain. It's not entirely clear what you consider a vulnerability—could you please describe it in more detail?

Hey, the comment says that the tokens are needed to be wrapped while they're not synced to actually be updated in the ATA

byshape

Comment fixed in https://github.com/1inch/solana-crosschain-protocol/pull/116

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.