SHERLOCK

Security Review For 1inch

Collaborative Audit Prepared For:	**1inch**
Lead Security Expert(s):	**0xeix**
	bin2chen
Date Audited:	**August 12 - August 16, 2025**
Final Commit:	**d0a59ab**

Introduction

This security review focuses on updates on 1inch's Cross-Chain swap.

Scope

Repository: 1inch/cross-chain-swap

Audited Commit: d0a59ab2c4b6be5c9769d5775769681873fcf162

Final Commit: d0a59ab2c4b6be5c9769d5775769681873fcf162

Files:

- contracts/BaseEscrowFactory.sol
- contracts/BaseEscrow.sol
- contracts/EscrowDst.sol
- contracts/EscrowFactoryContext.sol
- contracts/EscrowFactory.sol
- contracts/Escrow.sol
- contracts/EscrowSrc.sol
- contracts/interfaces/IBaseEscrow.sol
- contracts/interfaces/IEscrowDst.sol
- contracts/interfaces/IEscrowFactory.sol
- contracts/interfaces/IEscrow.sol
- contracts/interfaces/IEscrowSrc.sol
- contracts/interfaces/IMerkleStorageInvalidator.sol
- contracts/interfaces/IResolverExample.sol
- contracts/libraries/ImmutablesLib.sol
- contracts/libraries/ProxyHashLib.sol
- contracts/libraries/TimelocksLib.sol
- contracts/MerkleStorageInvalidator.sol
- contracts/zkSync/EscrowDstZkSync.sol
- contracts/zkSync/EscrowFactoryZkSync.sol
- contracts/zkSync/EscrowSrcZkSync.sol
- contracts/zkSync/EscrowZkSync.sol
- contracts/zkSync/MinimalProxyZkSync.sol

- contracts/zkSync/ZkSyncLib.sol

Final Commit Hash

d0a59ab2c4b6be5c9769d5775769681873fcf162

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
0	1	1

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: _withdraw() may be blocked by malicious integratorFeeRecipient

Source: https://github.com/sherlock-audit/2025-08-1inch-cross-chain-v1-1-0/issues/25

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

Vulnerability Detail

this version adds paying fees when `EscrowDst._withdraw()`

```
    function _withdraw(bytes32 secret, Immutables calldata immutables)
        internal
        onlyValidImmutables(immutables.hash())
        onlyValidSecret(secret, immutables.hashlock)
    {
        uint256 integratorFeeAmount = immutables.integratorFeeAmountCd();
        uint256 protocolFeeAmount = immutables.protocolFeeAmountCd();
        if (integratorFeeAmount > 0) {
@>>>          _uniTransfer(immutables.token.get(),
 ↪  immutables.integratorFeeRecipientCd().get(), integratorFeeAmount);
        }
        if (protocolFeeAmount > 0) {
@>>>          _uniTransfer(immutables.token.get(),
 ↪  immutables.protocolFeeRecipientCd().get(), protocolFeeAmount);
        }
        uint256 amount = immutables.amount - integratorFeeAmount -
         ↪  protocolFeeAmount;
        _uniTransfer(immutables.token.get(), immutables.maker.get(), amount);
        _ethTransfer(msg.sender, immutables.safetyDeposit);
        emit EscrowWithdrawal(secret);
    }
```

This leads to the risk that `withdraw()/publicWithdraw()` may not be executed. Examples:

1. `token == eth` , a malicious `integratorFeeRecipient` can revert in `receive()` to block `withdraw()`.
2. `integratorFeeRecipient` enter the token blacklist (e.g. USDC)

When `secret` is known, but `EscrowDst.withdraw()/publicWithdraw()` can't be executed, then `taker` can get token via `EscrowDst.cancel()` and also get funds from `EscrowSrc/EscrowDst`, `maker` will lose funds.

Impact

malicious `integratorFeeRecipient` + `taker` can steal `maker` funds

Code Snippet

https://github.com/sherlock-audit/2025-08-1inch-cross-chain-v1-1-0/blob/221c1233e647388a692a09048c287d87c966a004/cross-chain-swap/contracts/EscrowDst.sol#L87

Recommendation

`try/catch` pays for the `fees` method, if it fails, record:`feesBalance[recipient] = fees_amount`, the recipient can use another method to get it back again, thus ensuring that `_withdraw()` doesn't revert

Discussion

Sunnesoft

Hey! Thanks for the detailed review!

We attentively checked this potential risk and have next operational routines on our off-chain side: All integrators undergo KYC/KYB and must explicitly request access to the fees feature. Only approved integrators can set integratorFeeRecipient. If an integrator deploys malicious code (e.g., a reverting receive()), they would first harm their own users; in such cases we immediately revoke access and blacklist the integrator.

bin2chen66

Hey, thanks for the reply

Got it. This is a good way to minimize this risk. If possible for the next version, it is recommended to introduce a distribution mechanism to completely solve this risk in code.

lpetroulakis

The team acknowledges the potential issue. This issue doesn't warrant fixing at this time. It can be addressed in the next version of the code.

Issue L-1: Inconsistency when calculating the valid partial fill

Source: https://github.com/sherlock-audit/2025-08-1inch-cross-chain-v1-1-0/issues/26

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

Summary

The current implementation calculates the partial fill differently on different chains.

Vulnerability Detail

Let's take a look at the current implementation:

https://github.com/sherlock-audit/2025-08-1inch-cross-chain-v1-1-0/blob/main/cross-chain-swap/contracts/BaseEscrowFactory.sol#L208-228

```
function _isValidPartialFill(
     uint256 makingAmount,
     uint256 remainingMakingAmount,
     uint256 orderMakingAmount,
     uint256 partsAmount,
     uint256 validatedIndex
 ) internal pure returns (bool) {
     uint256 calculatedIndex = (orderMakingAmount - remainingMakingAmount +
      ↪  makingAmount - 1) * partsAmount / orderMakingAmount;

     if (remainingMakingAmount == makingAmount) {
         // If the order is filled to completion, a secret with index i + 1 must be
          ↪  used
         // where i is the index of the secret for the last part.
         return (calculatedIndex + 2 == validatedIndex);
     } else if (orderMakingAmount != remainingMakingAmount) {
         // Calculate the previous fill index only if this is not the first fill.
         uint256 prevCalculatedIndex = (orderMakingAmount - remainingMakingAmount -
          ↪  1) * partsAmount / orderMakingAmount;
         if (calculatedIndex == prevCalculatedIndex) return false;
     }

     return calculatedIndex + 1 == validatedIndex;
 }
```

According to the following comment, the formula has to be implemented like this

```
// If the order is filled to completion, a secret with index i + 1 must be used
```

```
      // where i is the index of the secret for the last part.
```

However, the current implementation adds 2 instead of 1 because of the following code:

https://github.com/sherlock-audit/2025-08-1inch-cross-chain-v1-1-0/blob/main/cross-chain-swap/contracts/MerkleStorageInvalidator.sol#L68

```
lastValidated[key] = ValidationData(takerData.idx + 1, takerData.secretHash);
```

This can be a problem because, for example, on the Solana chain, it's implemented according to the comment:

```
if remaining_making_amount == making_amount {


    // If the order is filled to completion, a secret with index i + 1 must be used
    // where i is the index of the secret for the last part.
    return calculated_index + 1 == validated_index;
```

Impact

Potential problems when interacting with the contracts on different chains due to the inconsistency of implementations.

Code Snippet

https://github.com/sherlock-audit/2025-08-1inch-cross-chain-v1-1-0/blob/main/cross-chain-swap/contracts/MerkleStorageInvalidator.sol#L68

Tool Used

Manual Review

Recommendation

Consider using a unified approach when validating indices.

Discussion

Sunnesoft

Thanks for the report!

_isValidPartialFill() intentionally varies by chain: EVM: indexing starts at 1 (index 0 is ambiguous). Solana: indexing starts at 0, as the implementation is driven by remaining_amount.

This function is evaluated only on the source chain and does not affect destination-chain behavior or settlement.

rodiontr

> Thanks for the report!
> _isValidPartialFill() intentionally varies by chain: EVM: indexing starts at 1 (index 0 is ambiguous). Solana: indexing starts at 0, as the implementation is driven by remaining_amount.
> This function is evaluated only on the source chain and does not affect destination-chain behavior or settlement.

got it, thanks for the explanation!

lpetroulakis

The team acknowledges the potential issue. This issue doesn't warrant fixing at this time.

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.