AstraSec

1inch
Cross-chain Fusion Resolver
Security Audit Report

December 13, 2024

Contents

1 Introduction

1.1 About Cross-chain Fusion Resolver

1inch Fusion+ swap introduces a two-party atomic swap mechanism, optimized for EVM-compatible chains with an option to execute a swap by a single party as an ordinary limit order. The audited cross-chain fusion resolver fulfills Fusion+ user orders for 1inch cross-chain protocol.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/1inch/cross-chain-resolver
- Commit id: 7dc13e9299b64a529defd4c8cf3acc17c572debf

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the cross-chain fusion resolver protocol. Throughout this audit, we identified a total of **3** issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won’t Do	Addressed
Critical	–	–	–	–
High	1	–	–	1
Medium	–	–	–	–
Low	1	–	–	1
Informational	1	–	–	1
Total	3	–	–	3

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to its detailed page, or scroll down to view all details in sequence.

~~H-1~~ Potential ERC-1271 Signature Replay in CrossChainResolver

~~L-1~~ Revisited Signer Validation in isValidSignature()

~~I-1~~ Magic Number Used in _prepareEscrow()

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and re-mediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [H-1] Potential ERC-1271 Signature Replay in CrossChainResolver

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
CrossChainResolver.sol	Business Logic	High	High	Addressed

The CrossChainResolver contract supports EIP-1271, with the isValidSignature() function verifying signatures. However, the function does not further process the input hash, omitting critical details such as the CrossChainResolver contract address or chain id. It only checks whether the signature is valid from one of the CrossChainResolver owners' perspective for the given hash.

This creates a potential signature replay vulnerability when the CrossChainResolver contract is used with other DeFi protocols, like Permit2. In particular, if two CrossChainResolver contracts share the same owner and both have granted token approvals to the Permit2 contract, a legitimate Permit2 signature from one CrossChainResolver could pass the verification of the other, undermining the protocol's security and design.

To mitigate this issue, we recommend appending essential information, such as the chain id and the CrossChainResolver address, to the input hash. This addition helps prevent the replay of signatures intended for one CrossChainResolver in another.

cross-chain-resolver - CrossChainResolver.sol

```
/**
 * @notice See {IERC1271-isValidSignature}.
 */
function isValidSignature(bytes32 hash, bytes calldata signature) external view override returns (bytes4 magicValue) {
    address signer = ECDSA.recover(hash, signature);
    if (signer == _OWNER1 || signer == _OWNER2 || signer == _OWNER3) magicValue = this.isValidSignature.selector;
}
```

Remediation Append essential information to the input hash used for signature verification to prevent potential signature replay.

3.3.2 [L-1] Revisited Signer Validation in isValidSignature()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
CrossChainResolver.sol	Business Logic	Medium	Low	Addressed

The CrossChainResolver contract is managed by three owners which are initialized in the constructor(). To support EIP-1271, it implements the isValidSignature() function to validate if a signature is signed by the contract owners. The signature validation is processed by recovering the signer's address from the input hash and signature (line 162) , confirming that the signer is one of the three owners (line 163).

However, if the input signature is illegitimate, it returns address(0) as the signer's address. In particular, if any of the owners is not properly initialized in the constructor(), an invalid signature could still pass the validation of the isValidSignature(). Therefore, we recommend enhancing the validation process to ensure that the recovered signer's address is never address(0).

cross-chain-resolver - CrossChainResolver.sol

```
158  /**
159   * @notice See {IERC1271-isValidSignature}.
160   */
161  function isValidSignature(bytes32 hash, bytes calldata signature) external view override returns (bytes4 magicValue) {
162      address signer = ECDSA.recover(hash, signature);
163      if (signer == _OWNER1 || signer == _OWNER2 || signer == _OWNER3) magicValue = this.isValidSignature.selector;
164  }
```

Remediation Enhance the validation process within the isValidSignature() function to ensure the recovered signer's address is never address(0).

3.3.3 [I-1] Magic Number Used in _prepareEscrow()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
CrossChainResolver.sol	Coding Practices	N/A	N/A	Addressed

In the CrossChainResolver contract, the _prepareEscrow() function is used to compute the source escrow address and set bit 251 for the input takerTraits (line 182). In the imported TakerTraitsLib library, bit 251 is designated by the constant variable _ARGS_HAS_TARGET, indicating that the first 20 bytes of args are considered the target address for transferring maker's funds.

For better coding practice, we recommend using the descriptive _ARGS_HAS_TARGET variable instead of the numeric 'magic number' 251 in the _prepareEscrow() function.

cross-chain-resolver - CrossChainResolver.sol

```
170 function _prepareEscrow(
171     IBaseEscrow.Immutables calldata immutables,
172     TakerTraits takerTraits,
173     bytes calldata args
174 ) private returns (TakerTraits updatedTakerTraits, bytes memory argsMem) {
175     IBaseEscrow.Immutables memory immutablesMem = immutables;
176     immutablesMem.timelocks = TimelocksLib.setDeployedAt(immutables.timelocks, block.timestamp);
177     address computed = _FACTORY.addressOfEscrowSrc(immutablesMem);
178     (bool success,) = address(computed).call{ value: immutablesMem.safetyDeposit }("");
179     if (!success) revert IBaseEscrow.NativeTokenSendingFailure();
180
181     // _ARGS_HAS_TARGET = 1 << 251
182     updatedTakerTraits = TakerTraits.wrap(TakerTraits.unwrap(takerTraits) | uint256(1 << 251));
183     argsMem = abi.encodePacked(computed, args);
184 }
```

Remediation Use the descriptive _ARGS_HAS_TARGET variable instead of the numeric 'magic number' 251 in the _prepareEscrow() function.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI