AstraSec

1inch Fee Charging Security Audit Report

January 28, 2025

Contents

1 Introduction

1.1 About 1inch Fee Charging

The 1inch Fee Charging project enhances platform functionality by extending the Limit Order and Fusion protocols. Key features include the implementation of fee collection directly from the taker’s amount, benefiting both integrators and the protocol itself. Additionally, the project introduces a KYC token, which ensures that the protocol is only accessible to KYC-verified takers and resolvers, promoting compliance and security. Lastly, the project focuses on settlement updates to improve transaction efficiency and overall platform performance.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/1inch/limit-order-protocol
- CommitID: a304ab7

  contracts/extensions/AmountGetterWithFee.sol
  contracts/extensions/AmountGetterBase.sol
  contracts/extensions/FeeTaker.sol
  contracts/libraries/MakerTraitsLib.sol

- https://github.com/1inch/limit-order-settlement/
- CommitID: 80950a9

  contracts/KycNFT.sol
  contracts/Settlement.sol
  contracts/SimpleSettlement.sol

- https://github.com/1inch/solidity-utils
- CommitID: 422cc3f

  contracts/libraries/AddressLib.sol
  contracts/libraries/SafeERC20.sol
  contracts/libraries/UniERC20.sol
  contracts/libraries/ECDSA.sol

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the 1inch Fee Charging 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	–	–	–	–
Medium	–	–	–	–
Low	1	–	–	1
Informational	2	1	–	1
Undetermined	–	–	–	–

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~L-1~~ Accommodation Of Non-ERC20-Compliant Tokens

I-1 Possible DoS Risk in KycNFT::transferFrom()

~~I-2~~ Redundant/Unnecessary Code Removal

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 [L-1] Accommodation Of Non-ERC20-Compliant Tokens

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
SafeERC20.sol	Business Logic	Low	Low	Addressed

Although there is a standard ERC-20 specification, many token contracts may not strictly follow this specification or may have additional functionalities beyond it. The following is an example using the safeTransfer() function in the SafeERC20 contract.

The function internally calls _makeCall(), which executes a low-level call to the specified ERC20 token contract. It transfers the tokens to the target address via a low-level call and ensures that on success, it either validates the existence of the contract code or checks that the returned data is a boolean true. If no data is returned, it verifies that the contract code exists; if data is returned, it ensures that the returned value is true. If the specified token is USDT on the TRON blockchain, there will be a compatibility issue with non-standard ERC20 tokens when this function is executed. This is due to the fact that the transfer() function in the TRON USDT contract always returns false, which causes the function execution to fail.

solidity-utils-master - SafeERC20.sol

```
function safeTransfer(
    IERC20 token,
    address to,
    uint256 value
) internal {
    if (!_makeCall(token, token.transfer.selector, to, value)) {
        revert SafeTransferFailed();
    }
}
```

Remediation To improve compatibility, the implementation of the safeTransfer() function needs to consider the special case of USDT on the TRON blockchain.

Response By Team This issue has been resolved as the team confirmed that 1inch does not support the TRON blockchain.

3.3.2 [I-1] Possible DoS Risk in KycNFT::transferFrom()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
KycNFT.sol	Security	N/A	N/A	Acknowledged

The transferFrom() function in the KycNFT contract allows the contract owner or a user with a valid signature from the owner to transfer a non-fungible token from its holder to a specified address. During our review of the KycNFT contract's code implementation, we identify a potential DoS risk with these two transferFrom() functions. Specifically, if the holder of the tokenId to be transferred burns the NFT by front-running the transaction, the execution of transferFrom() will fail.

limit-order-settlement-master - KycNFT.sol

```
/**
 * @notice Transfers a token to a specified address. Only the owner can call this function.
 * @param from The address to transfer the token from.
 * @param to The address to transfer the token to.
 * @param tokenId The ID of the token to be transferred.
 */
function transferFrom(address from, address to, uint256 tokenId) public override onlyOwner {
    _transfer(from, to, tokenId);
}

/**
 * @notice Transfers a token from account to another by token owner. This function using a valid owner's signature.
 * @param from The address to transfer the token from.
 * @param to The address to transfer the token to.
 * @param tokenId The ID of the token to be transferred.
 * @param signature The signature of the owner permitting the transfer.
 */
function transferFrom(address from, address to, uint256 tokenId, bytes calldata signature)
    public onlyOwnerSignature(to, tokenId, signature) {
    _transfer(from, to, tokenId);
}
```

Remediation To improve compatibility, if the tokenId to be transferred does not exist, directly mint a new token to the specified address.

Response By Team This token serves as an access token that only grants the right to act as a resolver. All token owners are users who have completed KYC/KYB, and if an owner burns his/her token, it is likely the token is no longer needed. The transferFrom() function is primarily used to allow a user to change their address, so the burn scenario does not present a significant concern.

3.3.3 [I-2] Redundant/Unnecessary Code Removal

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

The helper function _parseFeeData() provided by the AmountGetterWithFee contract is used to parse the extraData bytes to extract various fee-related values, including the integrator fee, integrator share, resolver fee, and whitelist discount. During the review of this function's code implementation, we notice that there are redundant revert checks for the integrator fee and resolver fee. The function checks if integratorFee and resolverFee exceed _BASE_1E5, but the revert conditions are unnecessary since these values are already constrained by the size of the bytes being read. The maximum value for uint16 should inherently avoid exceeding the limit, making the revert checks redundant.

limit-order-protocol-master - AmountGetterWithFee.sol

```
function _parseFeeData(
    bytes calldata extraData,
    address taker,
    function (bytes calldata, address) internal view returns (bool, bytes calldata) _isWhitelisted
) internal view returns (bool isWhitelisted, uint256 integratorFee, uint256 integratorShare,
                         uint256 resolverFee, bytes calldata tail) {
    unchecked {
        integratorFee = uint256(uint16(bytes2(extraData)));
        if (integratorFee > _BASE_1E5) revert InvalidIntegratorFee();
        integratorShare = uint256(uint8(bytes1(extraData[2:])));
        if (integratorShare > _BASE_1E2) revert InvalidIntegratorShare();
        resolverFee = uint256(uint16(bytes2(extraData[3:])));
        if (resolverFee > _BASE_1E5) revert InvalidResolverFee();
        uint256 whitelistDiscountNumerator = uint256(uint8(bytes1(extraData[5:])));
        if (whitelistDiscountNumerator > _BASE_1E2) revert InvalidWhitelistDiscountNumerator();
        (isWhitelisted, tail) = _isWhitelisted(extraData[6:], taker);
        if (isWhitelisted) {
            resolverFee = resolverFee * whitelistDiscountNumerator / _BASE_1E2;
        }
    }
}
```

Remediation Consider the removal of the redundant code with a simplified implementation.

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