Smart Contract Security Audit Report

1inch Aggregation Router v6 & Limit Order v4 (October 2023 update)

1. Contents

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 09/10/2023 to 20/10/2023.

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 diff between 6.0.0-prerelease-2 and master branches in the Aggregation Router repository and 4.0.0-prerelease-8 and master branches in the Limit Order Protocol repository. Initial review was done for the commit 230fe7e53ab10744eb1afa38dac5ddc888ca7b99 in Aggregation Router and da3e187f46885dfd234d9f630bd0afdf5c2ccfc0 in Limit Order Protocol and the re-testing was done for the commit XXX.

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:

- Protocol owner,
- Taker,
- Maker and their maker contracts

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious maker contract or submitting malicious data*	Contract owner Maker
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

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 several low severity security issues which have been fixed and re-tested in the course of the work.

The other suggestions included some best practices (see Security Process Improvement).

The 1inch team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of October 20, 2023.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Maker can replace IAmountGetter contract with CREATE2	limit-order-protocol/contracts/OrderLib.sol	Low	Not fixed
curveSwapCallback should be protected	1inch-contract/contracts/routers/UnoswapRouter.sol	Low	Not fixed
Lack of Chainlink oracle stale price check	limit-order-protocol/contracts/helpers/ChainlinkCalculator.sol	Low	Not fixed
Missing permit in _fillContractOrder	limit-order-protocol/contracts/OrderMixin.sol	Low	Not fixed
Missing 0x0 check on input addresses	contracts/OrderMixin.sol	Low	Not fixed

Issue	Contract	Risk Level	Status
swapSelector is not sufficiently validated	1inch-contract/contracts/routers/UnoswapRouter.sol	**Info**	Not fixed
fillContractOrder should not accept a malleable signature	limit-order-protocol/contracts/helpers/ETHOrders.sol	**Info**	Not fixed
Misleading comment	contracts/OrderMixin.sol	**Info**	Not fixed
Redundant code	contracts/OrderMixin.sol	**Info**	Not fixed

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. Maker can replace IAmountGetter contract with CREATE2

Risk Level: Low

Status: Not fixed

Contracts:

- limit-order-protocol/contracts/OrderLib.sol

Location: Lines: 157. Function: `isValidExtension`.

Description:

When an order is filled the `extension` structure can contain the `AmountGetter` address controlled by the maker, which is used to calculate maker and taker amounts. The taker may check the `AmountGetter` contract and simulate how the order is filled before the actual contract call to ensure its correct execution.

However, the maker can front-run a taker's transaction and replace the `AmountGetter` contract without changing its address, if it was created via CREATE2 → SELFDESTRUCT technique. The new replaced contract will return different values in `getMakingAmount` and `getTakingAmount` functions allowing the maker to steal almost all the taker's approved balance on `takerAsset`.

It's worth noting that the effective exploitation is only possible when the taker fills the order with the zero threshold.

Remediation:

Calculate `extcodehash` of the `AmountGetter` contract and include it into the `order` or `extension` structure. Check that the value wasn't changed, while filling the order. This way it will be signed and the `AmountGetter` contract's code cannot be changed.

References:

- https://rekt.news/tornado-gov-rekt/

5.2. curveSwapCallback should be protected

Risk Level: Low

Status: Not fixed

Contracts:

- 1inch-contract/contracts/routers/UnoswapRouter.sol

Location: Lines: 651. Function: `curveSwapCallback`.

Description:

The function `curveSwapCallback` allows to transfer any token from the AggregationRouterV6 contract. Although the docstring says that it "can be called by anyone assuming there are no tokens stored on this contract between transactions", it contradicts another function `rescueFunds` that allows to do the same operation but is protected with `onlyOwner`.

Remediation:

Add an access control check to `curveSwapCallback` to verify that it was called by a Curve pool.

5.3. Lack of Chainlink oracle stale price check

Risk Level: Low

Status: Not fixed

Contracts:

- limit-order-protocol/contracts/helpers/ChainlinkCalculator.sol

Location: Lines: 38,62,71,80. Functions: `getMakingAmount, getTakingAmount, doublePrice`.

Description:

`ChainlinkCalculator` contract utilises Chainlink oracles in order to calculate making and taking amounts. However when calling `latestRoundData()` function it ignores `updatedAt` parameter which is returned by this function.

```
(, int256 latestAnswer,,,) = oracle.latestRoundData();
```

Ignoring this parameter may lead to incorrect amount calculations with potentially stale price.

Remediation:

Consider processing `updatedAt` parameter and checking that it's not older than a set value, e.g 1 hour.

References:

- https://docs.chain.link/docs/historical-price-data/#historical-rounds

5.4. Missing permit in _fillContractOrder

Risk Level: Low

Status: Not fixed

Contracts:

- limit-order-protocol/contracts/OrderMixin.sol

Location: Lines: 250. Function: `_fillContractOrder`.

Description:

In `_fillContractOrder` there is a comment that suggests that a maker's permit should be executed on the first fill:

`*// Check signature and apply order permit only on the first fill*`

However, there is no such action in `_fillContractOrder`, only signature is verified.

Remediation:

During the first fill (`remainingMakingAmount == order.makingAmount`) in `_fillContractOrder()` the contract should call `tryPermit` as in `_fillOrder()`.

5.5. Missing 0x0 check on input addresses

Risk Level: Low

Status: Not fixed

Contracts:

- contracts/OrderMixin.sol

Location: Lines: 279. Function: `_fill`.

Description:

The `_fill()` function doesn't check that the `target` variable is not equal to zero. It's possible because the `_parseArgs()` function parses it from the user input.

Remediation:

Consider checking that input address is not zero.

References:

- https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

5.6. swapSelector is not sufficiently validated

Risk Level: Info

Status: Not fixed

Contracts:

- 1inch-contract/contracts/routers/UnoswapRouter.sol

Location: Lines: 565. Function: `_curfe`.

Description:

The `_curfe()` function allows to perform calls to any address with a user controlled selector via the dex argument. There are certain restrictions on the calldata:

- first two arguments will be masked with `0xFF` on the 568-569 lines
- the 5th param can be either a boolean `useEth` or current contract address.

```
contracts/routers/UnoswapRouter.sol:
  568:             mstore(add(ptr, 0x04), and(shr(_CURVE_FROM_TOKEN_OFFSET,
dex), _CURVE_FROM_TOKEN_MASK))
  569:             mstore(add(ptr, 0x24), and(shr(_CURVE_TO_TOKEN_OFFSET,
dex), _CURVE_TO_TOKEN_MASK))
    574:                 mstore(add(ptr, offset), useEth)
  579:                mstore(add(ptr, offset), address())
```

With these calldata limitations we found a way to craft such calldata that could possibly make the aggregation router call `Permit2.transferFrom(AllowanceTransferDetails[])` from the `_curfe` function. `AllowanceTransferDetails` structure looks like this:

```
struct AllowanceTransferDetails {
      // the owner of the token
      address from;
      // the recipient of the token
      address to;
      // the amount of the token
      uint160 amount;
      // the token to be transferred
      address token;
  }
```

The function `_curfe` has the following prototype:

```
function _curfe(
      address recipient,
      uint256 amount,
      uint256 minReturn,
      Address dex
)
```

The `transferFrom` call to Permit2 fits in the calldata built by `_curfe` the following way:

- the array offset and its size will be masked with `0xFF` and will fit into the first two arguments of the called function,
- the `amount` argument of `_curfe` will be `from` of the `AllowanceTransferDetails` structure,
- the `minReturn` argument will be `to` property of the `AllowanceTransferDetails` structure,
- the current address of the aggregation router will be `amount` of the `AllowanceTransferDetails`
- `recipient` argument of `_curfe` will be the `token` of `AllowanceTransferDetails`.

The only thing that prevents aggregation router from making this call is that Permit2 doesn't have any of `_CURVE_COINS_SELECTORS` selector or a fallback function in it which is necessary as `_curfe` calls `curveCoins()` on the target address.

```
contracts/routers/UnoswapRouter.sol:
  447:     // 0x87cb4f57 - base_coins(uint256)
  448:     // 0x23746eb8 - coins(int128)
  449:     // 0xc6610657 - coins(uint256)
  450:     // 0xb739953e - underlying_coins(int128)
  451:     // 0xb9947eb0 - underlying_coins(uint256)
  513:             function curveCoins(pool, selectorOffset, index) -> coin {
```

```
  514:                 mstore(0, _CURVE_COINS_SELECTORS)
  515:                 mstore(add(selectorOffset, 4), index)
  516:                 if iszero(staticcall(gas(), pool, selectorOffset, 0x24,
0, 0x20)) {
  517:                     reRevert()
  518:                 }
  519:                 coin := mload(0)
  520:             }
```

Remediation:

Extra validation should be performed on `swapSelector` variable extracted from the dex argument of the `_curfe` function as a user can choose any value. Consider restricting this variable to actual possible selectors found in Curve pools.

5.7. fillContractOrder should not accept a malleable signature

Risk Level: Info

Status: Not fixed

Contracts:

- limit-order-protocol/contracts/helpers/ETHOrders.sol

Location: Lines: `107`. Function: `isValidSignature`.

Description:

The function `_fillContractOrder` accepts a `signature` argument which is verified with `ECDSA.isValidSignature`. The implementation of `isValidSignature` is located in a dependency contract: https://github.com/1inch/solidity-utils/blob/master/contracts/libraries/ECDSA.sol. In this library the function `isValidSignature` follows ERC1271 and calls `isValidSignature` on the `signer` argument, i.e. on the maker contract. The maker contract in its turn calls `recoverOrIsValidSignature` on the ECDSA library, which is prone to signature malleability since it explicitly allows both 65-byte and compact 64-byte EIP-2098 signatures:

```
function recoverOrIsValidSignature(
      address signer,
      bytes32 hash,
      bytes calldata signature
  ) internal view returns (bool success) {
      if (signer == address(0)) return false;
```

```
      if ((signature.length == 64 || signature.length == 65) && recover(hash,
signature) == signer) {
          return true;
      }
      return isValidSignature(signer, hash, signature);
  }
```

However, this does not pose a security risk at the moment of the audit as this signature is not used as some kind of an invalidator.

Remediation:

As a best practice, consider restricting signature length to 65.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories/GHSA-4h98-2769-gh6h

5.8. Misleading comment

Risk Level: Info

Status: Not fixed

Contracts:

- contracts/OrderMixin.sol

Location: Lines: 461. Function: `_parseArgs`.

Description:

The comment before `_parseArgs()` function says that this function will revert if a taker permit is invalid but it won't because there is no permit in it.

```
contracts/OrderMixin.sol:
  461:       * @dev The function will revert if the taker permit is invalid.
```

Remediation:

Consider removing this comment.

5.9. Redundant code

Risk Level: Info

Status: Not fixed

Contracts:

- contracts/OrderMixin.sol

Location: Lines: 495. Function: `_parseArgs`.

Description:

The `args` variable is not used anywhere after it is truncated so this code can be removed:

```
contracts/OrderMixin.sol:
  495:             args = args[interactionLength:];
```

Remediation:

Consider removing redundant code.

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.