1inch Aggregation Router V6 & Limit Order Protocol V4

Date	April 2023
Auditors	Chingiz Mardanov, David Oz

1 Executive Summary

This report presents the results of our engagement with **1INCH** to review **Aggregation Router V6 and Limit Order Protocol V4**.

The review was conducted over 4 weeks, from **the 13th of March 2023** to **the 14th of April 2023**, by **Chingiz Mardanov** and **David Oz**. A total of 8 person-weeks were spent.

2 Scope

Our review focused on three different commits depending on the repo we were looking at:

- Aggregation Router V6: `b301c15f8c58d51b371cfd10baf445461d09d1b7`
- Solidity Utils: `42615efa5dca3ed43be565097ccb82a9c3e87273`
- Limit Order Protocol V4: `38eb988e9f496432e8eb12ef47ee134ceba89a40`

The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the **1INCH** team, we identified the following priorities for our review:

1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

3 System Overview

Since this engagement was on the new and updated versions of the `Aggregation Router` and `Limit Order Protocol` you can learn more about the **System Overview** of the present code in the previous report, which can be found here.

In this report, we would like to highlight some of the changes that very made in the present code compared to the previous version:

- Updated code now merged routers Uniswap V2 and Uniswap V3 into one.
- Updated router now also includes the logic to perform Curve swaps.
- Loops that were used to facilitate the swaps with multiple hops were now unrolled into sequential code for gas optimization.
- The Limit Order Protocol code was refactored and now unites regular and RFQ orders in one contract.
- Maker and Taker traits were introduced and made it easier to follow the limit order-filling logic.
- Both the aggregation router and the limit order protocol now support the permit2 functionality.

4 Recommendations

4.1 Memory-Safe flag is used on potentially memory unsafe operations.

Resolution

Remediated as per the 1inch team in 1inch/solidity-utils@ `90dcb3b` and 1inch/1inch-contract@ `e4c241c` by changing the location of where the call result is stored to a free memory pointer.

Description

Since developers are allowed to use yul to manually manage the memory during the execution, developers must follow guidelines of memory management or be extremely cautious when violating those. Solidity also expects the developer to mark any yul code compliant with standard memory management rules as `memory-safe` which is used widely across the code.

In reality, the analyzed code base uses this flag even in situations that are not necessarily memory safe. Take this code snippet for example:

1inch-utils/contracts/libraries/SafeERC20.sol:L293-L303

```
function safeWithdrawTo(IWETH weth, uint256 amount, address to) internal {
    safeWithdraw(weth, amount);
    if (to != address(this)) {
        assembly ("memory-safe") {  // solhint-disable-line no-inline-assembly
            if iszero(call(gas(), to, amount, 0, 0, 0, 0)) {
                returndatacopy(0, 0, returndatasize())
                revert(0, returndatasize())
            }
        }
    }
}
```

In this code, you can see that the revert message is stored in the scratch space at location 0. The reason why this code is not memory safe is that the revert message could occupy more than 64 bytes (if the fallback function with custom logic is encountered as the withdrawal destination), which would not be considered “accepted practice”. Despite the code reverting it is still considered unsafe memory usage. See docs for more information: https://docs.soliditylang.org/en/v0.8.17/assembly.html#memory-safety

Here are some other occurrences of the same issue:

1inch-utils/contracts/libraries/SafeERC20.sol:L281-L291

```
function safeWithdraw(IWETH weth, uint256 amount) internal {
    bytes4 selector = IWETH.withdraw.selector;
    assembly ("memory-safe") {  // solhint-disable-line no-inline-assembly
        mstore(0, selector)
        mstore(4, amount)
        if iszero(call(gas(), weth, 0, 0, 0x24, 0, 0)) {
            returndatacopy(0, 0, returndatasize())
            revert(0, returndatasize())
        }
    }
}
```

router/contracts/routers/UnoswapRouter.sol:L181-L194

```
function _unwrapWETH(address to, uint256 amount) private {
    assembly ("memory-safe") {  // solhint-disable-line no-inline-assembly
        mstore(0, _WETH_WITHDRAW_CALL_SELECTOR)
        mstore(4, amount)
        if iszero(call(gas(), _WETH, 0, 0, 0x24, 0, 0)) {
            returndatacopy(0, 0, returndatasize())
            revert(0, returndatasize())
        }
        if iszero(call(gas(), to, amount, 0, 0, 0, 0)) {
            returndatacopy(0, 0, returndatasize())
            revert(0, returndatasize())
        }
    }
}
```

There are other places with the potential for a similar issue, but for a more definitive conclusion, the called function would need to be analyzed. Here are some examples of those:

router/contracts/routers/UnoswapRouter.sol:L369-L372

```
function reRevert() {
    returndatacopy(0, 0, returndatasize())
    revert(0, returndatasize())
}
```

router/contracts/routers/UnoswapRouter.sol:L323-L326

```
if iszero(call(gas(), pool, 0, ptr, 0x0104, 0, 0x40)) {
    returndatacopy(0, 0, returndatasize())
    revert(0, returndatasize())
}
```

router/contracts/routers/UnoswapRouter.sol:L536-L539

```
function reRevert() {
    returndatacopy(0, 0, returndatasize())
    revert(0, returndatasize())
}
```

router/contracts/routers/UnoswapRouter.sol:L251-L253

```
if iszero(staticcall(gas(), pool, 0, 4, 0, 0x40)) {
    returndatacopy(0, 0, returndatasize())
    revert(0, returndatasize())
```

router/contracts/routers/UnoswapRouter.sol:L280-L283

```
if iszero(call(gas(), pool, 0, ptr, 0xa4, 0, 0)) {
    returndatacopy(0, 0, returndatasize())
    revert(0, returndatasize())
}
```

4.2 `SafeERC20` - Functions that are never used outside of the library should be declared as `private` instead of `internal` Won't Fix

Description

Some of the functions of `SafeERC20` are not used outside the context of the library, and thus should be declared as `private` instead of `internal`. Doing so would improve the readability of the code, as it would make it clearer which functions are intended to be used externally and which are only meant to be used internally within the library.

Examples

1inch-utils/contracts/libraries/SafeERC20.sol:L70-L75

```
function safeTransferFromPermit2(
    IERC20 token,
    address from,
    address to,
    uint160 amount
) internal {
```

1inch-utils/contracts/libraries/SafeERC20.sol:L151-L153

```
function tryPermit(IERC20 token, bytes calldata permit) internal returns(bool success) {
    return tryPermit(token, msg.sender, address(this), permit);
}
```

1inch-utils/contracts/libraries/SafeERC20.sol:L155

```
function tryPermit(IERC20 token, address owner, address spender, bytes calldata permit) internal returns(bool success) {
```

5 Aggregation Router

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major 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.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 `UnoswapRouter` - Calls to `rescueFunds` might be front-ran by attackers to claim excess funds

Medium Won't Fix

Description

`AggregationRouterV6.rescueFunds` is a function only callable by the owner of the contract used to rescue any excess funds left in the contract. However, funds that are locked in the contract can be claimed by attackers by using other code paths. Let's assume that the contract holds an excess balance in some ERC20 token denoted as `token`. A potential attacker can use a contract with an implemented `IUniswapV3Pool.token0.selector` function that will return the desired `token` address, then use it to call `UnoswapRouter.uniswapV3SwapCallback` to extract tokens in the amount of `amount0Delta`.

Examples

router/contracts/routers/UnoswapRouter.sol:L586

```
safeERC20(token, 0, add(emptyPtr, _TRANSFER_SELECTOR_OFFSET), 0x44, 0x20)
```

Recommendation

Consider restricting all code paths inside `uniswapV3SwapCallback` to the Uniswap v3 pool only.

5.2 WETH can be withdrawn from the router contract. Minor Won't Fix

Description

In addition to a more generic way of withdrawing any ERC20 token from the contract, there is an additional one that allows withdrawing wETH specifically.

Examples

router/contracts/routers/UnoswapRouter.sol:L81-L88

```
function _unoswapTo(address from, address to, IERC20 token, uint256 amount, uint256 minReturn, Address dex) private returns(uint
    if (dex.shouldUnwrapWeth()) {
        returnAmount = _unoswap(from, address(this), token, amount, minReturn, dex);
        _unwrapWETH(to, returnAmount);
    } else {
        returnAmount = _unoswap(from, to, token, amount, minReturn, dex);
    }
}
```

The attacker could attempt to swap worthless tokens but make sure the `returnAmount` is exactly the amount of the wETH stuck in the contract. In that case, you could also pass a flag `unwarpWeth()` even though you were not using wETH. As a result, the attacker could donate the worthless tokens in exchange for the wETH.

Recommendation

One could prevent this by checking if the swap destination token is wETH, but that would come at a gas cost. Given the low likelihood of this happening, we can not insist on this change.

6 Limit Order Protocol

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major 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.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 OrderMixin is not safe to ERC721 and ERC1155 without additional checks. Medium Won't Fix

Description

During the audit, we identified several ways to use the `limit-order-protocol` in a way that will not yield an outcome that is expected by the participants of the trade when non-fungible tokens are involved. The root issues for those inconsistencies are the getter interactions. By manipulating the `takingAmount` and `makingAmount` malicious actors could leave the opposite side with invalid tokens.

Let's take a look at some examples:

Examples

Consider a scenario where the maker places an order to sell 100 ERC1155 of id 17 for 100 ERC1155 of the same collection but id 5. This is a real case for trading in-game assets that are represented as ERC1155. Taker then passes amount 17 and `isMakingAmount` as true. In the `isMakingAmount == true` case maker gets to overwrite the taking amount.

limit-order-protocol/contracts/OrderMixin.sol:L271

```
if (takerTraits.isMakingAmount()) {
```

Let's assume the taker sets it to 4 this time. Then it is reasonable to expect the taker to pass 5 as the `threshold`. But during the threshold check the revert will not happen because 4 is less than the `threshold`.

limit-order-protocol/contracts/OrderMixin.sol:L279

```
if (takingAmount > threshold) revert TakingAmountTooHigh();
```

Considering that ERC1155 does not allow for approving tokens with specific IDs malicious maker could take the token that the taker did not want to sell.

The same can be done if the taker passes `isMakingAmount` as false. In this case, the maker will need to return 17 as makingAmount and will then be able to pass 4 as `takingAmount`.

If we were to generalize the cases above we can see that malicious maker could try and get any taker token with an ID smaller than originally offered because the check is designed around ERC20 and only check to make sure that the taker is not sending more than originally intended, but in the NFT case we also should check if we are sending less.

It is also worth mentioning that the maker is also in danger if they do not ensure that the order they place is not allowing partial fills. Otherwise, the malicious taker could pass a smaller id in the `amount` than the order has advertised and try to take a smaller token `id` from the maker.

It is important to mention that this is possible because ERC1155 does not allow for per-token approvals and users can only either approve all or none. The same logic can be done in some ERC721 tokens, but that is less prevalent.

After discussing this with 1INCH team they have shared that they already do ensure that before adding the order to the backend they check that if the order is trading NFTs they do not allow getters or partial fills. This should protect most of the users from this attack vector but since this system is public and permissionless it is still good to be aware of this possibility.

7 Solidity Utils

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major 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.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

7.1 Potentially unsafe uint160 conversion in SafeERC20 library. Minor

Resolution

Remediated as per the 1inch team in 1inch/solidity-utils@`4ecd734` by adding an if clause that reverts if values exceeding the `uint160` are used with permit2.

Description

After the addition of the permit2 support in the `SafeTransferUtils` a conversion of `uint256` to `uint160` had to be done. Unfortunately, there is no check for the overflow which could potentially be dangerous in the future. While in the case of DEX swaps, this will most likely be mitigated by DEX checks, the same can not be said about the `OrderMixin` code.

1inch-utils/contracts/libraries/SafeERC20.sol:L33

```
safeTransferFromPermit2(token, from, to, uint160(amount));
```

If `safeTransferFromUniversal` was used directly in the `OrderMixin` contract, that could have resulted in the loss of funds since while the order can advertise a large token amount, the actual transfer can end up being much smaller.

The reason this attack is not viable as of now is that `safeTransferFromUniversal` is not used and `transferFrom` of the PERMIT2 contract is called directly bypassing the manual `uint160(amount)` conversion. The amount passed to the `trasferFrom` of permit2 is checked automatically by the EVM in this case. So in the code snippet above even if the amount is over `uint160` the sub-cal on line 476 will return with `false`.

limit-order-protocol/contracts/OrderMixin.sol:L467-L479

```
function _callPermit2TransferFrom(address asset, address from, address to, uint256 amount) private returns(bool success) {
    bytes4 selector = IPermit2.transferFrom.selector;
    assembly ("memory-safe") { // solhint-disable-line no-inline-assembly
        let data := mload(0x40)
        mstore(data, selector)
        mstore(add(data, 0x04), from)
        mstore(add(data, 0x24), to)
        mstore(add(data, 0x44), amount)
        mstore(add(data, 0x64), asset)
        let status := call(gas(), _PERMIT2, 0, data, 0x84, 0x0, 0x20)
        success := and(status, or(iszero(returndatasize()), and(gt(returndatasize(), 31), eq(mload(0), 1))))
    }
}
```

Recommendation

Add a check that the uint256 that is being cast is in the bounds of uint160.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
1inch-utils/contracts/libraries/SafeERC20.sol	740756ee09699ee8f4b5a242006af769fef1c88f
limit-order-protocol/contracts/OrderMixin.sol	88fc8a636bb80bd755671004968f425e4c0e53d5
limit-order-protocol/contracts/OrderLib.sol	aeae5c0675aeb5802d45eddc484f039d8138eaf3
limit-order-protocol/contracts/LimitOrderProtocol.sol	461a463557c40cc3617e82b816ce4b2d2837fb57
limit-order-protocol/contracts/libraries/Errors.sol	ecaef783c9270807986bf82d50b7591b4a1a8069
limit-order-protocol/contracts/libraries/BitInvalidatorLib.sol	b1f8031784cc2c31d9a911a8e151a271a02c0375
limit-order-protocol/contracts/libraries/OffsetsLib.sol	b630d7fe4516cc17a4ae84ced080f27ec71547af
limit-order-protocol/contracts/libraries/ExtensionLib.sol	85cb15fc9abc4e6e7d099830f27f1904350338be
limit-order-protocol/contracts/libraries/RemainingInvalidatorLib.sol	c6a5d48ec23847fb696d6f034d01e1a216e6c0e5
limit-order-protocol/contracts/libraries/TakerTraitsLib.sol	5160804f787207f0f50d1c37bb1ffdf2fc74f0d3
limit-order-protocol/contracts/libraries/MakerTraitsLib.sol	9bc9cb0ba7d5c23a89acf8cdacfd08bf38080477
limit-order-protocol/contracts/helpers/AmountCalculator.sol	afc817262b5ecf4b97949cd8674e1bf8fde59f9b
limit-order-protocol/contracts/helpers/PredicateHelper.sol	32027b2730d99ad7430c63afd2a02662b441f20b
limit-order-protocol/contracts/helpers/SeriesEpochManager.sol	456fefa1e12368676df83c4f159658e73ec80b44
router/contracts/helpers/RouterErrors.sol	b9f53a2b8a7e9b9fdb3ab611e8184de953533e68
router/contracts/AggregationRouterV6.sol	f45810dd86f4f6c6b359e72f61179f985eba31d2
router/contracts/libs/ProtocolLib.sol	db518f3fe863fedc63d92a46b7d3ed79d1680110
router/contracts/routers/UnoswapRouter.sol	dedacaca56aa4ded0f2b5df142b60d14ce5de91a
router/contracts/routers/GenericRouter.sol	341887ed37d51ca0161e3a7936e9b02bda15818c
router/contracts/routers/ClipperRouter.sol	11bd4224d511bf78958803e731e3c345c4f915b4

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.