1inch Audit

Aggregation Router v5

June 2022

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the 1inch Aggregation Router v5 which upgraded from version v4. 1inch Aggregation Router facilitates transactions by utilizing a wide range of protocols, and the audited contracts allow users to exchange tokens using other exchanges benefiting from the best rates at the moment. First we will provide a summary of our discoveries, and then we will show the details of our findings.

Scope

The audited files are from the git repository located at https://github.com/1inch/1inch-contract . The audit is based on the commit `2f16d3888b9cd7513c288d0f195411501c9e79`.

The audited files are:

- `contracts/AggregationRouterV5.sol`: facilitates trading across multiple DEX.
- `contracts/GenericRouter.sol`: Base swapping contract.
- `contracts/UnoswapRouter.sol`: Facilitates swapping assets using Uniswap.
- `contracts/UnoswapV3Router.sol`: Facilitates swapping assets using Uniswap v3.
- `contracts/LimitOrderProtocolRFQ.sol`: fulfilling of RFQ orders.
- `helpers/EIP712Instance.sol`: Wrapper for OpenZeppelin's EIP712.
- `helpers/Errors.sol`: Error definitions.
- `helpers/NonceManager.sol`: Signature nonce generator.
- `contracts/WhiteListRegistrySimple.sol`: Simple whitelist implementation.
- `contracts/AggregationExecutorBase.sol`: base aggregator contract.
- `contracts/AggregationExecutorSimple.sol`: Aggregator contract with no access control.
- `contracts/AggregationExecutorProtected.sol`: Aggregator contract with signature-based access control.

- `contracts/AggregationExecutorWhitelist.sol`: Aggregator contract with whitelist-based access control.
- `solidity-utils/contracts/libraries/UniERC20.sol`: ERC20 wrapper that works on native eth.

The scope of the audit is limited to those files. No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Analyses

Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Summary of Findings

We found no critical issues, a medium issue and no minor issues. Also, one enhancement was proposed.

Security Issues

ID	Title	Severity	Status
ME-01	`MakeCall()` function allows Ether Transfer To Arbitrary Address	Medium	Acknowledged

Security Issues Found

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit can have four distinct statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk

Critical Severity Issues

No issues found.

Medium Severity Issues

ME-01 Functions At AggregationExecutorBase Allow Ether Transfer To Arbitrary Address

Location:

- `contracts/AggregationExecutorBase.sol:72,75,105,108`
- `contracts/AggregationExecutorSimple.sol`
- `contracts/AggregationExecutorProtected.sol`
- `contracts/AggregationExecutorWhitelist.sol`

Public functions `makeCall(), MakeCalls(), makeCallsMemory()` and `makeCallMemory()` of the abstract base contract `AggregationExecutorBase.sol` allow any external caller to send the contract's balance to any target. Here we can see an example:

```
    function makeCall(CallDescription calldata desc) public override {
…
        address target = address(uint160(desc.mandatoryGasLimitTarget));
…
        (status,) = target.call{ value: desc.value }(desc.data);
```

The contract AggregationExecutorSimple.sol inherit this contract and has no access controls, so any external address can call any of those public functions, however `AggregationExecutorProtected.sol` and `AggregationExecutorWhitelist.sol` also inherit this contract, but add access controls so you need a correct signature or a whitelist to move funds respectively.

However, any user can bypass the access control implemented in those child contracts by directly calling the `makeCall(), makeCalls()` , `makeCallsMemory()` or `makeCallMemory()` public functions inherited from the base contract.

Recommendation

Do not allow any external address to call the `makeCall(), MakeCalls(), makeCallsMemory()` or `makeCallMemory()` functions directly, for example making them internal instead of public.

Status
Acknowledged. The dev team acknowledges that this is possible but the risk is mitigated as the contract will only hold balance temporarily. Note: While this contract was not part of the original scope, it was audited as a dependency.

Minor Severity Issues

No issues found.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

Table

ID	Title	Status
EN-01	Unnecessary SafeMath.sol usage	Resolved

Details

EN-01 Unnecessary SafeMath.sol usage

Location:

- `solidity-utils/contracts/libraries/UniERC20.sol:53`

`SafeMath.sol` is no longer needed starting with Solidity 0.8 as the compiler now has built-in overflow checking.

Recommendation
Remove the `SafeMath.sol` import and perform the operation natively.

Status
Resolved. Changes on commit 4df9e7964e89b46757521ed9de54363df419eb97.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest for other stakeholders of the project, including users of the audited contracts, owners or project investors.

Centralization

The contract owner can retire any funds that the AggregationRouterV5 and the WhiteListRegistrySimple contracts may have. Additionally, the whitelist is also controlled just by the contract owner.

Reentrancy points

Additionally, many reentrancy points were identified, for example at `FlashbotsExtension.sol:34` and `GenericRouter.sol:143`, but in those cases the reentrancies do not affect the contract security.

Upgrades

There is no provision in any contract for upgrades, and making contracts upgradables will require refactoring of the code, due to the use of constructors.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

AggregationRouterV5

Owner

The owner of the contract can rescue funds and delete the contract.

WhitelistRegistrySimple

Owner

The owner of the contract can rescue funds and add/remove addresses from the white list.

Changelog

- 2022-06-15 – Initial report based on commit `2f16d3888b9cd7513c288d0f195411501c9e79`.
- 2022-06-22 – IFixes for issues reported on this report were checked on commit 4df9e7964e89b46757521ed9de54363df419eb97

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the 1Inch project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.