# Smart Contract Security Audit Report

## Hydro Protocol

SECBIT

**Dec 20, 2018**

# 1. Introduction

The Hydro Protocol contract is a decentralized exchange contract deployed on Ethereum. SECBIT Labs conducted an audit from Nov 26th, 2018 to Dec 5th, 2018, analyzing the contract from 3 aspects: **implementation bugs**, **logic flaws** and **risk assessment**. The audit shows that the Hydro Protocol contract has no critical security issue. Several issues were found in the first round of audit as shown in the following list (check Section 4 for details). SECBIT Labs provided suggestions for the fix and improvement. After the developing team optimized and updated the code, SECBIT Labs re-audited this project from Dec 11th, 2018 to Dec 14th, 2018. The latest version has addressed all found issues and introduced no additional issue, and is ready for release.

| Type | Description | Level | Status |
|---|---|---|---|
| Implementation | 4.3.1 Proxy.sol withdrawEther(): incorrect event name | Mid | Fixed |
| Implementation | 4.3.2 Proxy.sol validateMatchResult(): invalid verification | Mid | Confirmed |
| Testing | 4.3.3 match_test.js: one test case inconsistent with description | Mid | Fixed |
| Comment | 4.3.4 Exchange.sol: incomplete comment | Low | Fixed |
| Mechanism Design | 4.3.5 Exchange.sol: low entry/exit cost for relayer | Low | Confirmed |
| Implementation | 4.3.6 Exchange.sol: potential rounding errors from small token decimals | Low | Confirmed |
| Mechanism Design | 4.3.7 Issues on scalping and token ecosystem | High | Removed |
| Implementation | 4.3.8 Exchange.sol: recordHotDistribution failure would affect order matching | Mid | Removed |
| Mechanism Design | 4.3.9 HotDistribution.sol: HOT distribution is based on the number of trades rather than the amount of traded assets | Mid | Removed |

| | | | |
|---|---|---|---|
| Redundant Code | 4.3.10 Redundant code | Low | Optimized |
| Implementation | 4.3.11 LibDiscount.sol: call return value unhandled | Low | Confirmed |
| Potential Risk | 4.4.1 HotDistribution.sol: require abundant HOT balance | Low | Removed |
| Implementation | 4.4.2 LibWhitelist.sol: iterate array in unknown length | Low | Confirmed |

# 2. Contract Information

This section describes the basic information and the code structure of the contact.

### 2.1 Basic Information

The following list shows the basic information of the Hydro Protocol contract.

| Name | Hydro Protocol |
| --- | --- |
| Lines of Code | 1028 |
| Source | Hydro Git |
| Initial Git Commit | 4832964386c2c45e37635a1f9e140a04ac0cf592 |
| Final Git Commit | eebb72e7a2d2a28a12db4bc1c4b33a7170af00eb |
| Stage | In development |

### 2.2 File List

The following list shows source files in the Hydro Protocol contract:

| Name | Description |
| --- | --- |
| HybridExchange.sol | Core of the protocol to handle orders matching and cancelation |
| Proxy.sol | For actually settling the results of matching orders |
| interfaces/IERC20.sol | ERC20 interface |
| lib/EIP712.sol | EIP712 Ethereum typed structured data hashing and signing |
| lib/LibDiscount.sol | Library to handle fee discount calculation |
| lib/LibExchangeErrors.sol | Constant strings for error messages |

| | |
|---|---|
| lib/LibMath.sol | Library to deal with rounding calculation |
| lib/LibOrder.sol | Library for order struct and some useful helpers |
| lib/LibOwnable.sol | Ownable contract to provide authorization control |
| lib/LibRelayer.sol | For relayers to opt into/out incentive system or manage delegation |
| lib/LibSignature.sol | Library to validate order signature |
| lib/LibWhitelist.sol | Library for owner to manage whitelist |
| lib/SafeMath.sol | General SafeMath library |

# 3. Contract Analysis

This section analyzes the Hydro Protocol contract from three aspects: the type of the contract, the account permissions in the contract, and the functionality of the contract.

## 3.1 Contract Type

The Hydro Protocol contract is a set of contracts for decentralized exchanges of various tokens on Ethereum. Users do not need to register additional accounts, pre-transfer tokens or delegate the control of their token accounts to any centralized agency (*e.g.*, a centralized exchange). The contract is primarily responsible to validate and settle down orders on chain.

## 3.2 Account Permissions

There are several roles of accounts in Hydro Protocol listed as described below.

- `owner` and `non-owner`

  These 2 roles are checked and managed by `LibOwnable`.

  For `HybridExchange`, only `owner` can call `chamgeDiscountConfig()` to change the parameter `discountConfig`.

  For `Proxy`, only `owner` can call `addAddress()` and `removeAddress()` to manage `whitelist`.

  In addition, `owner` can call `renounceOwnership()` and `transferOwnership()` to discard and transfer its owner permissions. In practice, `owner` should be set to a multi-signature contract so as to improve the security of the entire system and avoid the single point failure.

- `relayer`, `taker`, `maker`

  These 3 roles are used in the order matching.

  No special permission is assigned to these 3 roles. `taker` and `maker` sign and send orders to `relayer` off-chain. `relayer` then matches orders according to prices, and finally submits matched orders to the Hydro Protocol contract for the on-chain settlement.

- `whitelist` and `non-whitelist`

  These 2 roles are checked and managed by `LibWhitelist`.

Only accounts in `whitelist` are permitted to call `transferFrom()`, `transferEther()` and `transferToken()` in `Proxy` to transfer assets in the order settlement. In order to secure the asset transfer, only `owner` can add trusted `HybridExchange` contracts to `whitelist`.

### 3.3 Functionality

The main contracts in Hydro Protocol are `HybridExchange` and `Proxy`.

- `HybridExchange`

  `HybridExchange` is the core contract, which is responsible for the order matching and cancelling. The order matching is performed in following steps.

  1. Validate signatures and the integrity of submitted orders.
  2. Update orders states according to the matching engine rules.
  3. Call `Proxy` contract to transfer tokens.

- `Proxy`

  `Proxy` is responsible to transfer tokens. In advance of exchange, users need to approve a proper amount of ERC20 tokens to `Proxy`, or save a proper amount of ETH via `depositEther()`. `Proxy` only allows `HybridExchange` in `whitelist` to call its functions to transfer assets. Therefore, its asset transfer interface is only used in the order settlement.

  The introduction of `Proxy` isolates `HybridExchange` from token contracts. Therefore, after the upgrade of `HybridExchange`, existing users do not need to redo the approve operations, which can save users' GAS and improve the user experience.

  In addition, Hydro Protocol provides a time lock mechanism, which allows existing users to cancel their approvals in a period after the upgrade of `HybridExchange`. The time lock is implemented along with the multi-signature contract:

  1. For the upgrade transaction, only after the amount of confirmations reaches a specified number, the upgrade transaction becomes ready.
  2. Meanwhile, a time lock is attached to the upgrade transaction. Only after the lock expires, the upgrade transaction is executed.

# 4. Audit Details

This section introduces the process and the complete content of the audit, and explains the discovered issues and risks in detail.

## 4.1 Audit Process

The audit strictly followed the audit specification of SECBIT Labs. We analyzed the project from aspects of implementation, business logic and potential risks. The process is composed of following four steps.

1. Each audit team reviews the wallet application independently.
2. Each audit team evaluates the vulnerabilities and potential risks independently.
3. Each audit team shares its audit results, and reviews results from other teams.
4. All audit teams coordinate with the audit leader to prepare the final audit report.

In the audit, several security analyzers and scanners were used, including

- SECBIT Labs internal tools: Adelaide checker, SF-checker, badmsg.sender
- 3rd open-source tools: Mythril, Slither, SmartCheck, Securify

The analyzing and scanning results were reviewed manually by audit teams. Besides, audit teams inspected and reviewed the contract line by line.

## 4.2 Audit Result

A complete list of 21 audit items and the corresponding results are listed below.

| Number | Classification | Result |
|---|---|---|
| 1 | All functions work correctly in normal cases | ✓ |
| 2 | No obvious bug (*e.g.* unhandled integer overflow/underflow) | ✓ |
| 3 | Can be compiled without warnings and errors | ✓ |
| 4 | Pass security analyzers and scanners without issues | ✓ |
| 5 | Compliant with security development standards | ✓ |

| | | |
|---|---|---|
| 6 | No risk in low level calls (`call`, `delegatecall`, `callcode`) and inline assembly | ✓ |
| 7 | No deprecated or outdated usage | ✓ |
| 8 | Clear implementation; explicit function visibility, variable types and Solidity version number | ✓ |
| 9 | No redundant code | ✓ |
| 10 | No potential risk from the manipulation of timestamp and network environment | ✓ |
| 11 | No issue in calling external contracts (*e.g.*, token contracts) | ✓ |
| 12 | No high GAS consuming operations in contract | ✓ |
| 13 | Clear and explicit business logic | ✓ |
| 14 | Implementation consistent with comments, whitepaper, *etc*. | ✓ |
| 15 | No logic not mentioned in design | ✓ |
| 16 | No ambiguity in logic and implementation | ✓ |
| 17 | No obvious flaw in mechanism design | ✓ |
| 18 | No fairness issue in business logic *wst*. game theory | ✓ |
| 19 | No risk threatening the project team | ✓ |
| 20 | No risk threatening normal users | ✓ |
| 21 | No long-term risk; capable to upgrade | ✓ |

## 4.3 Issues

### 4.3.1 Proxy.sol withdrawEther(): incorrect event name

| Type | Severity | Impact | Status |
|---|---|---|---|
| Implementation | Mid | False event emits | Fixed |

**Description**

[Proxy.sol#L41] `withdrawEther()` emits the `Deposit()` event after the successful execution. `Transfer()` event would be a better fit.

```
function withdrawEther(uint256 amount) public {
    balances[msg.sender] = balances[msg.sender].sub(amount);
    msg.sender.transfer(amount);
    emit Deposit(msg.sender, amount);
}
```

**Status**

Fixed by using the `Withdraw()` event instead.

### 4.3.2 Proxy.sol validateMatchResult(): invalid verification

| Type | Severity | Impact | Status |
| --- | --- | --- | --- |
| Implementation | Mid | Invalid order validation | Confirmed |

**Description**

[Exchange.sol#L244] `validateMatchResult()` checks the final match result, which provides an additional layer of security protection. In detail, `totalMatch` records the current result, which is always smaller than `takerOrder` if orders are matched separately. In such case, the validation does not take effects.

```
function validateMatchResult(LightOrder memory takerOrder,
TotalMatchResult memory totalMatch) internal pure {
    if (isSell(takerOrder.data)) {
        // make sure doen't sell more base tokens
        require(totalMatch.baseTokenFilledAmount <=
takerOrder.baseTokenAmount, TAKER_SELL_BASE_EXCEEDED);
    } else {
        // make sure doen't cost more quote tokens
        require(totalMatch.quoteTokenFilledAmount <=
takerOrder.quoteTokenAmount, TAKER_MARKET_BUY_QUOTE_EXCEEDED);

        if (!isMarketOrder(takerOrder.data)) {
            // if there is a price improvement, make sure the taker
doesn't buy more base tokens than he(she) plans to
            require(totalMatch.baseTokenFilledAmount <=
takerOrder.baseTokenAmount, TAKER_LIMIT_BUY_BASE_EXCEEDED);
        }
    }
```

```
}
```

**Status**

Confirmed. This function may have this issue in the above circumstance, but it works for orders that are not separated. Specially, it intends to be an additional layer of protection and not cover all cases, so no change is made now.

### 4.3.3 match_test.js: one test case inconsistent with description

| Type | Severity | Impact | Status |
|---|---|---|---|
| Testing | Mid | Incomplete test case | Fixed |

**Description**

[match_test.js#L748] This case is described as `taker buy(market), maker full match`, but it actually tests the `limit buy` order.

```
takerOrderParams: {
    trader: u1,
    relayer,
    version: 1,
    side: 'buy',
    type: 'limit',
    expiredAtSeconds: 3500000000,
    asMakerFeeRate: 1000,
    asTakerFeeRate: 5000,
    baseTokenAmount: toWei('8424.22'),
    quoteTokenAmount: toWei('318.5197582'),
    gasTokenAmount: toWei('0.1')
},
```

**Status**

Fixed.

### 4.3.4 Exchange.sol: incomplete comment

| Type | Severity | Impact | Status |
|---|---|---|---|
| Comment | Low | Inaccurate comment | Fixed |

**Description**

[Exchange.sol#L240] The comment states this code `update filled amount of limit taker Order`. However, the implementation updates `market order` as well and the value is `quoteTokenFilledAmount`. It is recommended to specify the additional behavior explicitly in the comment.

```
// update filled amount of limit taker Order
filled[takerOrderInfo.orderHash] = takerOrderInfo.filledAmount;
```

**Status**

Fixed.

### 4.3.5 Exchange.sol: low entry/exit cost for relayer

| Type | Severity | Impact | Status |
|---|---|---|---|
| Mechanism Design | Low | Profit | Confirmed |

**Description**

[Exchange.sol#L211] `Exchange.sol` determines whether a relayer is joined by `isParticipantRelayer`. However, the entry and exit cost the relayer little, and can affect the fee discounts and the HOT distribution. Such design may hurt the profits of certain participants.

**Result**

Confirmed. The design has already considered the above issue. Relayers are free to exit, and HOT holders are free to choose relayers. No change is necessary now.

### 4.3.6 Exchange.sol: potential rounding errors from small token decimals

| Type | Severity | Impact | Status |
|---|---|---|---|
| Implementation | Low | Fee cost | Confirmed |

**Description**

[Exchange.sol#L361] The integer division in Solidity has rounding errors, which may produce a smaller result. Consider a case where the fee rate is 0.1%: when `quoteTokenFilledAmount` < 1000, the resulting `makerFee` may be smaller than the accurate result by 0.1%. The smaller the token decimals is, the more it will be impacted. However, `quoteToken` used in the following fee calculation is usually the

stable token or WETH, which has large decimals and suffers little from rounding errors.

```
// maker fee will be reduced, but still >= 0
uint256 makerFeeRate = getMakerFeeRate(makerOrder.trader,
makerRawFeeRate.sub(rebateRate), isParticipantRelayer);
result.makerFee =
result.quoteTokenFilledAmount.mul(makerFeeRate).div(feeRateBase.mul(d
iscountRateBase));
result.makerRebate = 0;
```

**Status**

Confirmed. The developing team has already considered this issue. The decimals of quoteToken is usually large, so the impact of rounding errors here is negligible. The Hydro Protocol documentation will notify relayers to avoid the issue of rounding errors when choosing `quoteToken`.

### 4.3.7 Issues on scalping and token ecosystem

| Type | Severity | Impact | Status |
| --- | --- | --- | --- |
| Mechanism Design | High | Token ecosystem | Removed |

**Description**

[Exchange.sol#L377] Relayers can get HOT with a relatively low GAS by submitting a large number of orders. When relayers, takers and makers are the same or in the same party, those submissions contribute little to the liquidity (which is different from the centralized exchanges). When the price of HOT is high, such scalping operations may impact the HOT token ecosystem a lot.

**Status**

The latest version has removed the code related to HotDistribution.

### 4.3.8 Exchange.sol: recordHotDistribution failure would affect order matching

| Type | Severity | Impact | Status |
| --- | --- | --- | --- |
| Implementation | Mid | Order matching | Removed |

**Description**

[Exchange.sol#L495] calls `record(address[])` of `HotDistribution`. If `id <= 0 || id > periodsCount`, it will return `false` and may revert the normal order matching.

```
if (result == 0) {
    revert(RECORD_HOT_DISTRIBUTION_ERROR);
}
```

**Status**

The latest version has removed the code related to HotDistribution.

#### 4.3.9 HotDistribution.sol: HOT distribution is based on numbers of trades rather than the amount of traded assets

| Type | Severity | Impact | Status |
| --- | --- | --- | --- |
| Mechanism Design | Mid | Token ecosystem | Removed |

**Description**

[HotDistribution.sol#L102] In the token economic model, the HOT distribution is based on the number of trades rather than trading amount. Such design may encourage users to separate their orders for more HOT distributions, and lower the scalping cost.

**Status**

The latest version has removed the code related to HotDistribution.

#### 4.3.10 Redundant code

| Type | Severity | Impact | Status |
| --- | --- | --- | --- |
| Redundant Code | Low | Gas consumption | Optimized |

**Description**

The two `require` checks in [Proxy.sol#L68] and [SafeMath.sol#L20] are redundant because of the use of safe math in the subsequent code, though they may be useful for debug.

```
function transferEther(address from, address to, uint256 value)
    internal
```

```
    onlyAddressInWhitelist
{
    require(balances[from] >= value, "ETHER_TRANSFER_FROM_FAILED");
    balances[from] = balances[from].sub(value);
    balances[to] = balances[to].add(value);
    emit Transfer(from, to, value);
}

function div(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b > 0, "DIVIDING_ERROR");
    uint256 c = a / b;
    return c;
}
```

**Status**

Optimized.

#### 4.3.11 LibDiscount.sol: call return value unhandled

| Type | Severity | Impact | Status |
| --- | --- | --- | --- |
| Implementation | Low | Potentially unsafe external calls | Confirmed |

**Description**

[LibDiscount.sol#L51] The `0` return value of `call` is not handled. However, the risk is low, because the callee is trusted.

```
result := call(gas, hotToken, 0, 0, 36, 0, 32)
result := mload(0)
```

**Status**

Confirmed. The callee is trusted, so no check is needed. The explanation has been added in the comment.

### 4.4 Potential Risks

SECBIT Labs found the following risks after assessing the Hydro Protocol contract.

#### 4.4.1 HotDistribution.sol: require abundant HOT balance

| Type | Severity | Impact | Status |
| --- | --- | --- | --- |

| | | | |
|---|---|---|---|
| Potential Risk | Low | External dependency | Removed |

**Description**

[HotDistribution.sol#L82] If the contract does not have sufficient HOT, user's claim operation will fail. However, users are still able to redo the claim operation.

```
if (amount > 0) {
    require(IERC20(hotTokenAddress).transfer(msg.sender, amount),
"CLAIM_HOT_TOKEN_ERROR");
}
```

**Status**

The latest version has removed the code related to HotDistribution.

### 4.4.2 LibWhitelist.sol: iterate array in unknown length

| Type | Severity | Impact | Status |
|---|---|---|---|
| Implementation | Low | Array Iteration | Confirmed |

**Description**

[LibWhitelist.sol#L49] The owner should take care the length of array `allAddresses`. When it becomes large, iterating the array will result in a high GAS consumption.

```
for(uint i = 0; i < allAddresses.length; i++){
    if(allAddresses[i] == adr) {
        allAddresses[i] = allAddresses[allAddresses.length - 1];
        allAddresses.length -= 1;
        break;
    }
}
```

**Status**

Confirmed. The administrator shall take care and restrict the length of `allAddresses` in practice.

# 5. Conclusion

The Hydro Protocol contract completely implements all key functions of a decentralized exchange protocol. In the audit, SECBIT Labs found several issues and possible optimizations, and proposed suggestions for the fix and improvement accordingly. The latest version of the Hydro Protocol contract has addressed all those issues, introduced no additional one, and satisfied the standard for the secure release. Specially, the code quality of the Hydro Protocol contract is high, which includes the consistent and good function names, clear code structures, complete comments and high test coverage. The design of the exchange protocol is neat and efficient, which contains the compression of order parameters and optimizations for the amount of transfers. In addition, the developing team is professional and responsive. They always fixed issues quickly, which is impressive.

# Disclaimer

The smart contract audit service from SECBIT Labs assesses the contract's correctness, security and performability from the aspects of code quality, logic design and potential risks. The report is provided "as is", without any warranties about the code practicability, business model, management system's applicability and anything related to the contract adaptation. This audit report is not to be taken as an endorsement of the platform, team, company or investment.

# APPENDIX

### Vulnerability/Risk Level Classification

| Severity | Description |
|---|---|
| High | Severely damage the contract's integrity and allow attackers to steal ethers and tokens, or lock ethers inside the contract. |
| Mid | Damage contract's security under given conditions and cause impairment of benefit for stakeholders. |
| Low | Cause no actual impairment to contract. |
| Info | Relevant to practice or rationality of the contract, could possibly bring risks. |

**SECBIT Labs is devoted to construct a common-consensus, reliable and ordered blockchain economic entity.**


 http://www.secbit.io

 audit@secbit.io

 @secbit_io