智能合约安全审计报告

Hydro Protocol

SECBIT

Dec 14, 2018

安比（SECBIT）实验室致力于解决区块链全生态的安全问题，提供区块链全生态的安全服务。作为中国信息通信研究院区块链安全技术组成员，参与编写区块链安全白皮书和参与制定区块链安全审计规范。

安比（SECBIT）实验室智能合约审计从合约的技术实现、业务逻辑、接口规范、Gas 优化、发行风险等维度，由两组专业审计人员分别独立进行安全审计，审计过程借助于安比实验室研发的一系列形式化验证、语义分析等工具进行扫描检测，力求尽可能全面地解决所有安全问题。

1. 综述

Hydro Protocol 是用于部署在以太坊上的去中心化交易合约。安比（SECBIT）实验室于 2018 年 11 月 26 日至 2018 年 12 月 05 日对合约进行审计。审计过程从**实现漏洞**、**逻辑漏洞**和**风险评估**三个维度对合约进行分析。审计结果表明，Hydro Protocol 合约并未包含致命的安全漏洞，安比（SECBIT）实验室给出了如下代码修复或优化建议项（详见第4章节）。Hydro Protocol 开发团队对代码进行了优化更新。安比（SECBIT）实验室于 2018 年 12 月 11 日至 2018 年 12 月 14 日进行复审。经审计，更新后的代码解决了此前发现的所有问题，并未引入新的安全风险，达到了安全发布标准。

风险类型	描述	风险级别	最终状态
实现漏洞	4.3.1 Proxy.sol withdrawEther() 事件名称写错	中	已修复
实现漏洞	4.3.2 Proxy.sol validateMatchResult() 校验无效	中	已确认
测试用例	4.3.3 match_test.js 一处测试用例与描述不相符	中	已修复
注释与文档	4.3.4 Exchange.sol 一处注释描述不完整	低	已修复
机制设计	4.3.5 Exchange.sol relayer 加入/退出成本很低	低	已确认
实现漏洞	4.3.6 Exchange.sol Token decimals 较小时会有 rounding error	低	已确认
机制设计	4.3.7 刷单成本及 Token 生态问题	高	已移除
实现漏洞	4.3.8 Exchange.sol recordHotDistribution 失败的情况而影响撮合	中	已移除
机制设计	4.3.9 HotDistribution.sol HOT 分发根据交易笔数而不是交易金额	中	已移除
冗余代码	4.3.10 存在部分冗余代码	低	已改进
实现漏洞	4.3.11 LibDiscount.sol 存在一处 call 返回	低	已确认

	值未处理		
风险提示	4.4.1 HotDistribution.sol 合约需要有足量 HOT 余额	低	已移除
实现漏洞	4.4.2 LibWhitelist.sol 存在一处未知长度数组遍历	低	已确认

2. 合约信息

该部分描述了合约的基本信息和代码组成。

2.1 基本信息

以下列表展示了 Hydro Protocol 的基本信息：

名称	**Hydro Protocol**
行数	1028
文件来源	Hydro Git
Git Commit （初审）	4832964386c2c45e37635a1f9e140a04ac0cf592
Git Commit （复审）	eebb72e7a2d2a28a12db4bc1c4b33a7170af00eb
合约阶段	开发阶段

2.2 合约列表

以下展示了 Hydro Protocol 项目包含的合约列表：

合约名称	**描述**
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. 合约分析

该部分描述了合约代码的详细分析内容，从合约类型，合约账户分类和合约功能分析三部分来进行说明。

3.1 合约类型

Hydro Protocol 是去中心化交易智能合约，用于在以太坊平台上实现各类 Token 的去中心化撮合交易。参与交易的用户无需预先注册或充值，无需托管账户。智能合约主要负责订单的撮合校验和链上结算。

3.2 合约账户分类

Hydro Protocol 中有多种角色账户。按照对应关系列举如下。

- owner 和 非 owner

  这两种角色由 LibOwnable 控制区分。

  对于 HybridExchange 合约，仅 owner 账户可调用 changeDiscountConfig() 函数来更改折扣配置项 discountConfig。

  对于 Proxy 合约，仅 owner 账户可调用 addAddress() 或 removeAddress() 函数来添加或删除 whitelist 名单。

  此外，owner 账户还可分别调用 renounceOwnership() 或 transferOwnership() 来放弃或转移 owner 权限。为了整个系统的安全性和避险发生单点失效，owner 账户在实际使用中应该设置为多重签名合约。

- relayer、taker、maker

  这三种角色应用于订单撮合功能。

  合约中不另做限制，三种角色分别代表订单撮合过程中的中继者、吃单者和挂单者。用户（吃单者或挂单者）将订单签名后链下提交给中继者，由中继者链下进行价格匹配，最终提交合约进行结算。

- whitelist 和 非 whitelist

  这两种角色由 LibWhitelist 控制区分。

  仅 whitelist 账户可以调用 Proxy 合约中的 transferFrom()、transferEther() 和 transferToken() 函数，用于在订单撮合结算的过程中转移参与各方资产。在实际使用过程中，为了确保用户资产能够安全转移，whitelist 账户必须只能由 owner 设置为可信的 HybridExchange 合约。

3.3 功能分析

Hydro Protocol 的主合约分别是 HybridExchange 和 Proxy。

- HybridExchange 合约

  HybridExchange 合约是协议核心，负责撮合校验和取消订单。订单匹配的主要步骤如下：

  1. 校验签名，验证订单正确性
  2. 根据撮合引擎计算并更新合约状态
  3. 调用转移 Token 的 Proxy 合约进行结算

- Proxy 合约

`Proxy` 合约负责转移 Token，用户交易前把相关 ERC20 Token approve 一定额度给 Proxy 合约或提前调用 `depositEther()` 存入以太币。`Proxy` 合约内有权限校验，只有 `HybridExchange` 合约可以调用实际转账接口。因此该部分转移用户资产的功能，仅可用于结算订单。

`Proxy` 合约的引入使得 `HybridExchange` 和 Token 合约有效隔离。因此，在升级 `HybridExchange` 核心协议合约后，老用户无需重新 approve，可有效节省 Gas 并提高用户体验。另外，Hydro Protocol 还有时间锁机制，可在升级 `HybridExchange` 后，给用户一定时间取消 approve。时间锁的具体实现利用了多重签名合约：对于交易提案，当确认数达到合约规定值时，交易进入待执行状态，并加上时间锁。到达时间锁规定时间后，交易才可以执行。

4. 审计详情

该部分描述合约审计流程和详细结果，并对发现的问题（实现漏洞，代码优化和逻辑漏洞），风险点和附加提示项进行详细的说明。

4.1 审计过程

本次审计工作，严格按照安比（SECBIT）实验室审计流程规范执行，从技术实现，业务逻辑以及合约发行风险三个维度进行全面分析。审计流程大致分为四个步骤:

- 各审计小组对合约代码进行逐行分析，根据合约审计内容要求进行审计
- 各审计小组对合约漏洞和风险进行评估
- 审计小组之间交换审计结果，并对审计结果进行逐一审查和确认
- 审计小组配合审计负责人生成审计报告

4.2 审计结果

本次审计首先经过安比（SECBIT）实验室推出的分析工具 adelaide、 sf-checker 和 badmsg.sender（内部版本）扫描，再利用开源安全分析工具 Mythril、Slither、SmartCheck 以及 Securify 检查，检查结果由审计小组成员详细确认。审计小组成员对合约源码进行逐行检查、评估，汇总审计结果。审计内容总结为如下 21 大项。

编号	分类	结果
1	合约各功能可以正常执行	通过
2	合约代码不存在明显的漏洞（如整数溢出等）	通过
3	能够通过编译并且没有任何警告输出	通过

4	合约代码能够通过常见检测工具检测，并无明显漏洞	通过
5	符合安全开发规范	通过
6	底层调用（call, delegatecall, callcode）或内联汇编的操作不存在安全隐患	通过
7	代码中不包含已过期或被废弃的用法	通过
8	代码实现清晰明确，函数可见性定义明确，变量数据类型定义明确，合约版本号明确	通过
9	不存在冗余代码	通过
10	不存在受时间和外部网络环境影响的隐患	通过
11	调用外部合约符合规范，如 Token 合约	通过
12	合约不存在明显的 Gas 损耗	通过
13	业务逻辑实现清晰明确	通过
14	代码实现逻辑与注释、项目白皮书等资料保持一致	通过
15	代码不存在设计意图中未提及的逻辑	通过
16	业务逻辑实现不存在疑义	通过
17	机制设计不存在明显的缺陷	通过
18	博弈论角度评估业务逻辑不存在公平性问题	通过
19	不存在危及相关项目方利益的明确风险	通过
20	不存在危及参与合约的普通用户利益的明确风险	通过
21	项目发行不存在长远隐患，且预留升级空间	通过

4.3 问题列表

4.3.1 Proxy.sol withdrawEther() 事件名称写错

风险类型	风险级别	影响点	状态
实现漏洞	中	虚假事件触发	已修复

问题描述

[Proxy.sol#L41] `withdrawEther()` 函数操作成功后触发 `Deposit()` 事件，不合理。触发 `Transfer()` 事件更合常理。

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

后续状态

已修复，现已改为使用更合适的 `Withdraw()` 事件。

4.3.2 Proxy.sol validateMatchResult() 校验无效

风险类型	风险级别	影响点	状态
实现漏洞	中	校验订单	已确认

问题描述

[Exchange.sol#L244] `validateMatchResult()` 函数用于最终校验订单撮合结果，起增加一层保障的作用。`totalMatch` 记录的是本次撮合的结果，因此在实际操作中，只要订单分批撮合，`totalMatch` 就肯定小于 `takerOrder`。这种场景下起不到校验订单撮合情况的作用。

```
    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);
            }
        }
    }
```

后续状态

已确认。在分批撮合场景下的确可能存在该问题。不过对于非分批撮合订单，该检查有效。该功能属于额外的一层保护，不追求所有场景下适用，因此暂不做改动。

4.3.3 match_test.js 一处测试用例与描述不相符

风险类型	风险级别	影响点	状态
测试用例	中	测试用例不完整	已修复

问题描述

[match_test.js#L748] 该用例描述为 `taker buy(market), maker full match`，实际测试的是 `limit buy` 订单。

```
            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')
            },
```

后续状态

已修复。

4.3.4 Exchange.sol 一处注释描述不完整

风险类型	风险级别	影响点	状态
注释与文档	低	注释不准确	已修复

问题描述

[Exchange.sol#L240] 注释描述称 `update filled amount of limit taker Order`，实际上对于 `market order` 也更新了，并且值是 `quoteTokenFilledAmount`。注释中值得强调指出。

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

后续状态

已修复。

4.3.5 Exchange.sol relayer 加入/退出成本很低，但会影响交易者手续费折扣

风险类型	风险级别	影响点	状态
机制设计	低	多方收益	已确认

问题描述

[Exchange.sol#L211] 通过 isParticipantRelayer 校验 relayer 是否加入。但 relayer 加入/退出成本很低（LibRelayer.sol），而且会影响交易者手续费折扣和 HOT 分发，这种设计是否会存在问题?

后续状态

已确认。relayer 有自由退出的权利，HOT 持有人，可以自由选择 relayer。协议设计已考虑此问题，因此暂不做改动。

4.3.6 Exchange.sol Token decimals 较小时会有 rounding error

风险类型	风险级别	影响点	状态
实现漏洞	低	手续费计算	已确认

问题描述

[Exchange.sol#L361] 此处存在除法，理论上也会有 rounding error。假设手续费为 0.1%(1E4)，当 quoteTokenFilledAmount < 1000，会有 0.1 % 的误差，这种极端情况下，对于 decimals 小的 token 可能问题比较大。但归根结底 decimals 太小的币，根据百分比算 fee 始终会有问题。而且这里是 quoteToken，通常是稳定币或 WETH 等币种，decimals 都比较正常。因此风险较小。

```
            // 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;
```

后续状态

已确认。开发团队已考虑过该问题，quoteToken 的 decimals 通常较大，因此正常场景下不会受此问题影响。协议文档中将会提示 relayer 在选择 quoteToken 时注意规避 rounding error 问题。

4.3.7 刷单成本及 Token 生态问题

风险类型	风险级别	影响点	状态
机制设计	高	Token 生态	已移除

问题描述

[Exchange.sol#L377] 目前 relayer 刷单（交易挖矿，换取 HOT）成本较低。当 relayer taker maker 均为一方时，直接调用合约撮合提交订单，对市场流通量无任何帮助（与中心化交易所不一样），消耗少量 gas 费用，换取 HOT。当 HOT 市场价值较高时，影响很大。Token 生态可能会面临问题。

后续状态

新版本中已暂时移除 HotDistribution 相关代码。

4.3.8 Exchange.sol recordHotDistribution 失败的情况，会影响订单撮合

风险类型	风险级别	影响点	状态
实现漏洞	中	撮合交易	已移除

问题描述

[Exchange.sol#L495] 调用 `HotDistribution` 合约的 `record(address[])` 接口，实际程序有分支会 `return false`，具体场景是 `id <= 0 || id > periodsCount`，需要当心此时无法正常撮合交易。

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

后续状态

新版本中已暂时移除 HotDistribution 相关代码。

4.3.9 HotDistribution.sol HOT 分发根据交易笔数而不是交易金额

风险类型	风险级别	影响点	状态
机制设计	中	Token 生态	已移除

问题描述

[HotDistribution.sol#L102] Token 经济模型里，HOT 分发根据交易笔数而不是交易金额，是否合理和健康？这个模式下一方面会鼓励用户分多笔订单交易，另一方面会进一步降低刷单成本。

后续状态

新版本中已暂时移除 HotDistribution 相关代码。

4.3.10 存在部分冗余代码

风险类型	风险级别	影响点	状态
冗余代码	低	Gas 消耗	已改进

问题描述

[Proxy.sol#L68] 和 [SafeMath.sol#L20] 这两处 require 代码校验没有必要。当然，为了方便 debug 可以保留。

```
    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;
    }
```

后续状态

已改进。

4.3.11 LibDiscount.sol 存在一处 call 返回值未处理

风险类型	风险级别	影响点	状态
实现漏洞	低	外部调用	已确认

问题描述

[LibDiscount.sol#L51] 此处存在 call 返回值未处理（result 为 0，异常未处理），不过此处调用的是受信任的合约，故而风险较低。

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

后续状态

已确认。因调用信任合约，此处无需处理，已添加注释说明。

4.4 风险提示

安比（SECBIT）实验室在对 Hydro Protocol 合约风险进行评估以后，指出合约存在如下风险项：

4.4.1 HotDistribution.sol 合约需要有足量 HOT 余额

风险类型	风险级别	影响点	状态
风险提示	低	外部依赖	已移除

问题描述

[HotDistribution.sol#L82] 需保证本合约有大量 HOT，否则用户会 claim 失败，不过用户仍可重新 claim。

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

后续状态

新版本中已暂时移除 HotDistribution 相关代码。

4.4.2 LibWhitelist.sol 存在一处未知长度数组遍历

风险类型	风险级别	影响点	状态
实现漏洞	低	数组循环	已确认

问题描述

[LibWhitelist.sol#L49] 此处应当心 allAddresses 数组变得很大，不过 owner 使用上注意即可。

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

后续状态

已确认。管理员实际使用时会注意控制 allAddresses 数组长度。

5. 结论

Hydro Protocol 完整地实现了去中心化交易协议的关键功能。安比（SECBIT）实验室在对 Hydro Protocol 合约进行审计分析后，发现了部分代码缺陷和可优化项，并提出了对应的修复及优化建议，上文均已给出具体的分析说明。经审计，Hydro Protocol 开发团队更新后的代码解决了此前发现的所有问题，并未引入新的安全风险，达到了安全发布标准。此外值得一提的是，安比（SECBIT）实验室认为，Hydro Protocol 项目代码质量较高，函数命名规范、代码结构清晰、注释完整，拥有较高的测试用例覆盖率。整个交易协议设计精简高效，尤其是对订单参数的压缩精简和结算转账次数的优化。同时，Hydro Protocol 开发团队高效、专业，第一时间回应并修复所发现的问题，令人印象深刻。

免责声明

SECBIT 智能合约安全审计从合约代码质量、合约逻辑设计和合约发行风险等方面对合约的正确性、安全性、可执行性进行审计，但不做任何和代码的适用性、商业模式和管理制度的适用性及其他与合约适用性相关的承诺。本报告为技术信息文件，不作为投资指导，也不为相关交易背书。

附录

漏洞风险级别介绍

风险级别	风险描述
高	可以严重损害合约完整性的缺陷，能够允许攻击者盗取以太币及Token，或者把以太币锁死在合约里等缺陷。
中	在一定限制条件下能够损害合约安全的缺陷，造成某些参与方利益损失的缺陷。
低	并未对合约安全造成实质损害的缺陷。
提示	不会带来直接的风险，但与合约安全实践或合约合理性建议有关的信息。

安比（SECBIT）实验室致力于参与共建共识、可信、有序的区块链经济体。

https://secbit.io
audit@secbit.io
@secbit_io