JustSwap

智能合约安全审计报告

2020-08-25

1. 概要

慢雾安全团队于 2020 年 08 月 17 日，收到 JustSwap 团队对 JustSwap 系统安全审计的申请，根据项目特点慢雾安全团队制定如下审计方案。

慢雾安全团队将采用“白盒为主，黑灰为辅”的策略，以最贴近真实攻击的方式，对项目进行安全审计。

慢雾科技 DApp 项目测试方法：

黑盒测试	站在外部从攻击者角度进行安全测试。
灰盒测试	通过脚本工具对代码模块进行安全测试，观察内部运行状态，挖掘弱点。
白盒测试	基于项目的源代码，进行脆弱性分析和漏洞挖掘。

慢雾科技 DApp 漏洞风险等级：

严重漏洞	严重漏洞会对项目的安全造成重大影响，强烈建议修复严重漏洞。
高危漏洞	高危漏洞会影响项目的正常运行，强烈建议修复高危漏洞。
中危漏洞	中危漏洞会影响项目的运行，建议修复中危漏洞。
低危漏洞	低危漏洞可能在特定场景中会影响项目的业务操作，建议项目方自行评估和考虑这些问题是否需要修复。
弱点	理论上存在安全隐患，但工程上极难复现。
增强建议	编码或架构存在更好的实践方法。

2. 审计方法

慢雾安全团队智能合约安全审计流程包含两个步骤:

- 使用开源或内部自动化分析的工具对合约代码中常见的安全漏洞进行扫描和测试。
- 人工审计代码的安全问题，通过人工分析合约代码，发现代码中潜在的安全问题。

如下是合约代码审计过程中我们会重点审查的漏洞列表:

（其他未知安全漏洞不包含在本次审计责任范围）

- 重入攻击
- 重放攻击
- 重排攻击
- 短地址攻击
- 拒绝服务攻击
- 交易顺序依赖
- 条件竞争攻击
- 权限控制攻击
- 整数上溢/下溢攻击
- 时间戳依赖攻击
- Gas 使用，Gas 限制和循环
- 冗余的回调函数
- 不安全的接口使用
- 函数状态变量的显式可见性
- 逻辑缺陷
- 未声明的存储指针
- 算术精度误差
- tx.origin 身份验证
- 假充值漏洞
- 变量覆盖

3. 项目概述

3.1 项目描述

JustSwap 是一个基于 TRON 的交换协议，可以用于 TRC20 tokens 的交换。

项目官网地址：

https://justswap.io/

审计合约文件：

项目源代码

justswap.tar.gz:

MD5: be4b075dc236d2f614b06d6da419603b

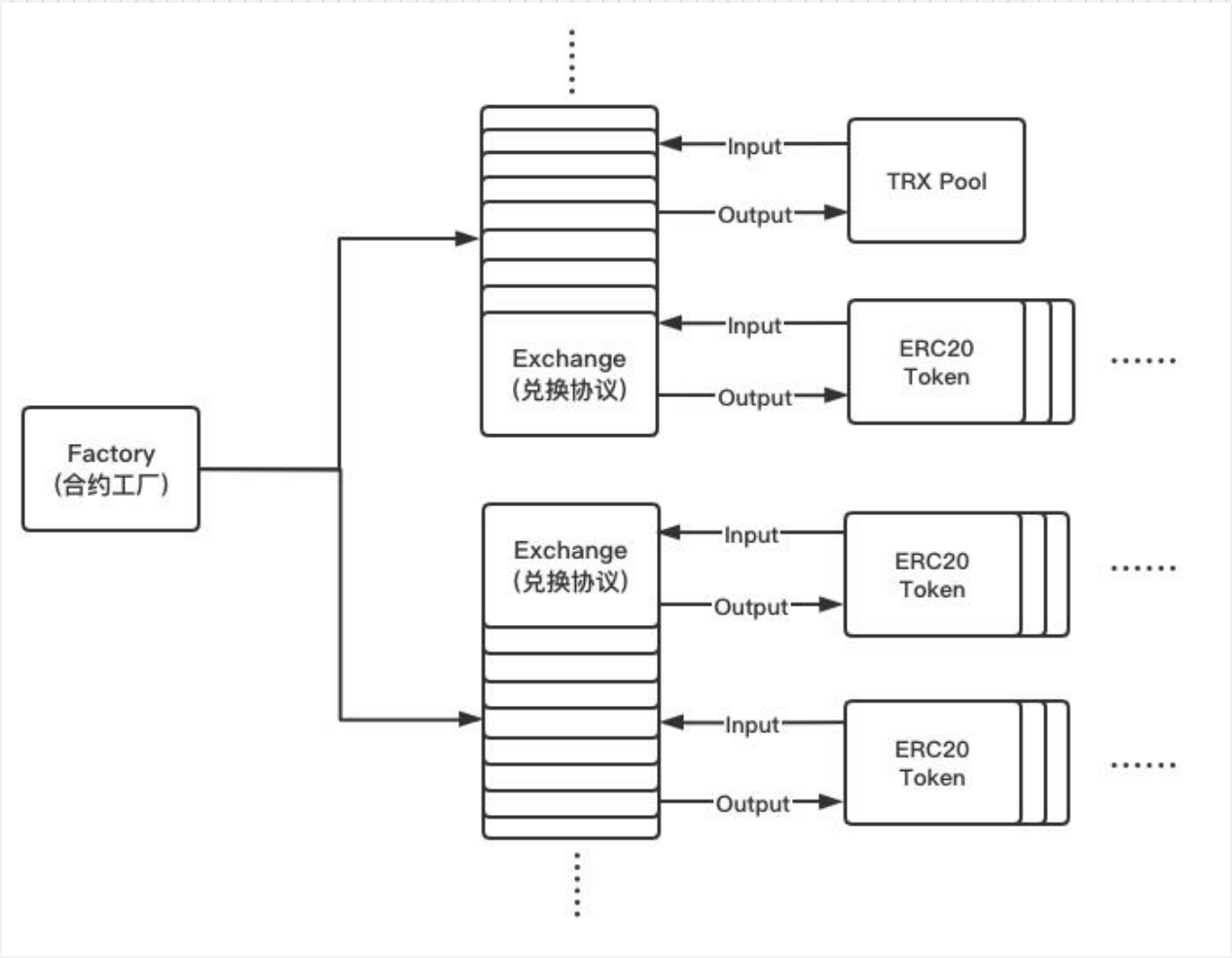
3.2 项目结构

justswap:

```
.
├── README.md
├── interfaces
│   ├── IJustswapExchange.sol
│   ├── IJustswapFactory.sol
│   └── ITRC20.sol
├── justswap
│   ├── JustswapExchange.sol
│   └── JustswapFactory.sol
├── tokens
│   └── TRC20.sol
└── utils
    ├── ReentrancyGuard.sol
    ├── SafeMath.sol
    └── TransferHelper.sol
```

3.3 合约架构

JustSwap DApp 主要分为两个部分，分别为合约工厂与代币兑换部分。其中 JustswapFactory 合约负责实现为每个 TRC20 代币创建一个独立的交易合约。JustswapExchange 合约负责实现提供流动性池进行代币兑换、手续费处理以及自定义资金池的功能。每个交易合约都会与一个 TRC20 token 关联，并且带有一个 TRX 和这个 TRC20 token 的流动性池，以实现 TRX 与 token 的兑换功能以及 token 与 token 之间的兑换功能，兑换过程中产生的手续费存入流动性池中。合约整体架构图如下：



4. 代码分析

4.1 主要合约文件及哈希

No	File Name	SHA-1 Hash
1	JustswapExchange.sol	64c33c94f379890d580c5971ec44073cd42c7e7c
2	TRC20.sol	c02bb96b3a004f3b82db83a80b1c53abe3426992
3	SafeMath.sol	50f1e0e4fd6bc2002e4111221991382f18e0fef4
4	ITRC20.sol	c130889347ff735dbe86ea35af3799a67350e92c
5	IJustswapFactory.sol	fd2b8e7b53515bf242059bd1b47cdcdc9b2a0227
6	IJustswapExchange.sol	3b8d8cbef22e2f418f50ff1e61cf6140359ab6f8
7	ReentrancyGuard.sol	8170dd07eee4eb2402ad65f3323eadb0dc2d8709
8	TransferHelper.sol	d6fe29a53d7f142e6a296e198658a28bf5ce8945
9	JustswapFactory.sol	84d90357c979f5f81991551986033ccf6589a729

4.2 主要合约函数可见性分析

Contract Name	Function Name	Visibility
JustswapExchange	Implementation	TRC20, ReentrancyGuard
	setup	Public

	getInputPrice	Public
	getOutputPrice	Public
	trxToTokenInput	Private
	trxToTokenSwapInput	Public
	trxToTokenTransferInput	Public
	trxToTokenOutput	Private
	trxToTokenSwapOutput	Public
	trxToTokenTransferOutput	Public
	tokenToTrxInput	Private
	tokenToTrxSwapInput	Public
	tokenToTrxTransferInput	Public
	tokenToTrxOutput	Private
	tokenToTrxSwapOutput	Public
	tokenToTrxTransferOutput	Public
	tokenToTokenInput	Private
	tokenToTokenSwapInput	Public
	tokenToTokenTransferInput	Public
	tokenToTokenOutput	Private
	tokenToTokenSwapOutput	Public
	tokenToTokenTransferOutput	Public

	tokenToExchangeSwapInput	Public
	tokenToExchangeTransferInput	Public
	tokenToExchangeSwapOutput	Public
	tokenToExchangeTransferOutput	Public
	getTrxToTokenInputPrice	Public
	getTrxToTokenOutputPrice	Public
	getTokenToTrxInputPrice	Public
	getTokenToTrxOutputPrice	Public
	tokenAddress	Public
	factoryAddress	Public
	addLiquidity	Public
	removeLiquidity	Public
TRC20	Implementation	——
	totalSupply	Public
	balanceOf	Public
	allowance	Public
	transfer	Public
	approve	Public
	transferFrom	Public
	increaseAllowance	Public

	decreaseAllowance	Public
	_transfer	Internal
	_mint	Internal
	_burn	Internal
	_approve	Internal
	_burnFrom	Internal
SafeMath	Library	——
	mul	Internal
	div	Internal
	sub	Internal
	add	Internal
	mod	Internal
ITRC20	Interface	——
	transfer	External
	approve	External
	transferFrom	External
	totalSupply	External
	balanceOf	External
	allowance	External
IJustswapFactory	Interface	——

	initializeFactory	External
	createExchange	External
	getExchange	External
	getToken	External
	getTokenWihId	External
IJustswapExchange	Interface	——
	getInputPrice	External
	trxToTokenTransferInput	External
	trxToTokenSwapOutput	External
	trxToTokenTransferOutput	External
	tokenToTrxSwapInput	External
	tokenToTrxTransferInput	External
	tokenToTrxSwapOutput	External
	tokenToTrxTransferOutput	External
	tokenToTokenSwapInput	External
	tokenToTokenTransferInput	External
	tokenToTokenSwapOutput	External
	tokenToTokenTransferOutput	External
	tokenToExchangeSwapInput	External
	tokenToExchangeTransferInput	External

	tokenToExchangeSwapOutput	External
	tokenToExchangeTransferOutput	External
	getTrxToTokenInputPrice	External
	getTrxToTokenOutputPrice	External
	getTokenToTrxInputPrice	External
	getTokenToTrxOutputPrice	External
	tokenAddress	External
	factoryAddress	External
	addLiquidity	External
	removeLiquidity	External
ReentrancyGuard	Implementation	——
TransferHelper	Library	——
	safeApprove	Internal
	safeTransfer	Internal
	safeTransferFrom	Internal
JustswapFactory	Implementation	——
	initializeFactory	Public
	createExchange	Public
	getExchange	Public
	getToken	Public

	getTokenWithId	Public

4.3 代码审计详情

4.3.1 Liquidity 无法移除风险

在 addLiquidity 函数中使用了`address(token).safeTransferFrom(msg.sender, address(this), token_amount)`，而在 removeLiquidity 函数中使用了`address(token).safeTransfer(msg.sender, token_amount)`，两者不一致可能导致 Liquidity 无法移除的风险。例如：某合约的 transferFrom 的返回值符合 TIP20 标准中定义的关于 TRC20 代币的返回值规范，而 transfer 的返回值不符合 TIP20 标准中定义的关于 TRC20 代币的返回值规范，则可能造成 addLiquidity 可以成功，但 removeLiquidity 无法成功的问题。

代码位置： JustswapExchange.sol 文件第 631 与 660 行

```
    function addLiquidity(uint256 min_liquidity, uint256 max_tokens, uint256 deadline) public payable nonReentrant returns (uint256) {
    require(deadline > block.timestamp && max_tokens > 0 && msg.value > 0, 'JustExchange#addLiquidity: INVALID_ARGUMENT');
    uint256 total_liquidity = _totalSupply;

    if (total_liquidity > 0) {
      require(min_liquidity > 0, "min_liquidity must greater than 0");
      uint256 trx_reserve = address(this).balance.sub(msg.value);
      uint256 token_reserve = token.balanceOf(address(this));
      uint256 token_amount = (msg.value.mul(token_reserve).div(trx_reserve)).add(1);
      uint256 liquidity_minted = msg.value.mul(total_liquidity).div(trx_reserve);

      require(max_tokens >= token_amount && liquidity_minted >= min_liquidity, "max tokens not meet or liquidity_minted not meet min_liquidity");
      _balances[msg.sender] = _balances[msg.sender].add(liquidity_minted);
      _totalSupply = total_liquidity.add(liquidity_minted);

      require(address(token).safeTransferFrom(msg.sender, address(this), token_amount), "transfer failed");
      emit AddLiquidity(msg.sender, msg.value, token_amount);
```

```
        emit Snapshot(msg.sender,address(this).balance,token.balanceOf(address(this)));
        emit Transfer(address(0), msg.sender, liquidity_minted);
        return liquidity_minted;
    } else {
        require(address(factory) != address(0) && address(token) != address(0) && msg.value >= 10_000_000,
"INVALID_VALUE");
        require(factory.getExchange(address(token)) == address(this), "token address not meet exchange");
        uint256 token_amount = max_tokens;
        uint256 initial_liquidity = address(this).balance;
        _totalSupply = initial_liquidity;
        _balances[msg.sender] = initial_liquidity;

        require(address(token).safeTransferFrom(msg.sender, address(this), token_amount), "tranfer failed");
        emit AddLiquidity(msg.sender, msg.value, token_amount);
        emit Snapshot(msg.sender,address(this).balance,token.balanceOf(address(this)));
        emit Transfer(address(0), msg.sender, initial_liquidity);
        return initial_liquidity;
    }
}
```

```
  function removeLiquidity(uint256 amount, uint256 min_trx, uint256 min_tokens, uint256 deadline) public nonReentrant
returns (uint256, uint256) {
    require(amount > 0 && deadline > block.timestamp && min_trx > 0 && min_tokens > 0, "illegal input parameters");
    uint256 total_liquidity = _totalSupply;
    require(total_liquidity > 0, "total_liquidity must greater than 0");
    uint256 token_reserve = token.balanceOf(address(this));
    uint256 trx_amount = amount.mul(address(this).balance) / total_liquidity;
    uint256 token_amount = amount.mul(token_reserve) / total_liquidity;
    require(trx_amount >= min_trx && token_amount >= min_tokens, "min_token or min_trx not meet");

    _balances[msg.sender] = _balances[msg.sender].sub(amount);
    _totalSupply = total_liquidity.sub(amount);
    msg.sender.transfer(trx_amount);

    require(address(token).safeTransfer(msg.sender, token_amount), "transfer failed");
    emit RemoveLiquidity(msg.sender, trx_amount, token_amount);
    emit Snapshot(msg.sender,address(this).balance,token.balanceOf(address(this)));
    emit Transfer(msg.sender, address(0), amount);
    return (trx_amount, token_amount);
  }
}
```

修复情况：经与项目方确认，TRC20 规范要求 transfer 和 transferFrom 符合指定格式。Justswap 仅支持通过 TronScan 审核的 TRC20 代币。所有不符合 TRC20 规范的 Token，JustSwap 不支持交易。

4.3.2 部分代码冗余

JustswapFactory 合约中存在 initializeFactory 函数，用来对合约进行初始化。初始化时需传入非 0 地址的 exchangeTemplate，并在 createExchange 函数中进行创建交易合约前对其进行判断，但之后 exchangeTemplate 并未用到，直接使用`new JustswapExchange()`创建交易合约。因此 initializeFactory 函数与 createExchange 函数中对 exchangeTemplate 的检查是冗余的。TRC20 合约中_mint 函数其可见性为 internal 且未被其他合约调用，_burnFrom 函数其可见性也为 internal 且未被其他合约调用，此代码冗余。TransferHelper 合约中 safeApprove 函数其可见性为 internal 且未被其他合约调用，此代码冗余。

代码位置：JustswapFactory.sol 文件第 25 至 29 行与第 33 行，TRC20.sol 文件第 139 行与 183 行，TransferHelper.sol 文件第 6 行。

```
function initializeFactory(address template) public {
    require(exchangeTemplate == address(0), "exchangeTemplate already set");
    require(template != address(0), "illegal template");
    exchangeTemplate = template;
}
```

```
function createExchange(address token) public returns (address) {
    require(token != address(0), "illegal token");
    require(exchangeTemplate != address(0), "exchangeTemplate not set");
    require(token_to_exchange[token] == address(0), "exchange already created");
    JustswapExchange exchange = new JustswapExchange();
    exchange.setup(token);
    token_to_exchange[token] = address(exchange);
    exchange_to_token[address(exchange)] = token;
    uint256 token_id = tokenCount + 1;
    tokenCount = token_id;
    id_to_token[token_id] = token;
```

```
    emit NewExchange(token, address(exchange));
    return address(exchange);
}
```

```
function _mint(address account, uint256 value) internal {
    require(account != address(0));

    _totalSupply = _totalSupply.add(value);
    _balances[account] = _balances[account].add(value);
    emit Transfer(address(0), account, value);
}
```

```
function _burnFrom(address account, uint256 value) internal {
    _burn(account, value);
    _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
}
```

```
    function safeApprove(address token, address to, uint value) internal returns (bool){
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
        return (success && (data.length == 0 || abi.decode(data, (bool))));
    }
```

修复情况：经与项目方确认，对业务无影响，代码不做修改。

5. 审计结果

5.1 低危漏洞

- 移除流动性池设计缺陷

5.2 增强建议

- 部分代码冗余

5.3 总结

审计结论 ： 通过

审计编号 ：0X002008250002

审计时间 ：2020 年 08 月 25 日

审计团队 ： 慢雾安全团队

审计总结 ： 本次审计发现 2 个安全问题，经过沟通反馈确认审计过程中发现的风险均在可承受范围内。

6. 声明

慢雾仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，慢雾无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向慢雾提供的文件和资料(简称“已提供资料”)。慢雾假设：已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，慢雾对由此而导致的损失和不利影响不承担任何责任。

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号