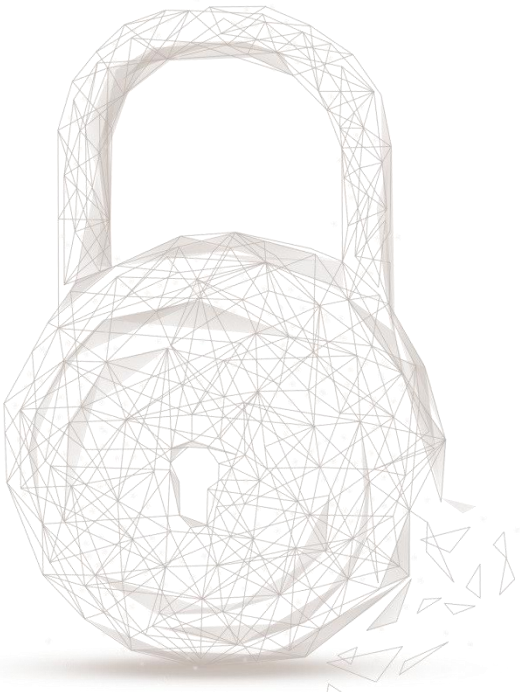
成都链安
BEOSIN

智能合约安全审计报告



成都链安 BEOSIN

审计编号：202101121119

报告查询名称：FC-LiquidityProvider

审计合约信息：

合约名称	审计合约地址	审计合约链接地址
LiquidityProvider	TSfVcQCYTV3WvpcFNGSSxmwU9ic98VQMCS	https://tronscan.org/#/contract/TSfVcQCYTV3WvpcFNGSSxmwU9ic98VQMCS/code

合约审计开始日期：2021.01.08

合约审计完成日期：2021.01.12

审计结果：通过

审计团队：成都链安科技有限公司

审计类型及结果：

序号	审计类型	审计子项	审计结果
1	代码规范审计	编译器版本安全审计	通过
		弃用项审计	通过
		冗余代码审计	通过
		require/assert 使用审计	通过
		gas 消耗审计	通过
2	通用漏洞审计	整型溢出审计	通过
		重入攻击审计	通过
		伪随机数生成审计	通过
		交易顺序依赖审计	通过
		拒绝服务攻击审计	通过
		函数调用权限审计	通过
		call/delegatecall 安全审计	通过
		返回值安全审计	通过

		tx.origin 使用安全审计	通过
		重放攻击审计	通过
		变量覆盖审计	通过
3	业务审计	业务逻辑审计	通过
		业务实现审计	通过

备注：审计意见及建议请见代码注释。

审计结果说明：

本公司采用形式化验证、静态分析、动态分析、典型案例测试和人工审核的方式对FC-LiquidityProvider项目智能合约代码规范性、安全性以及业务逻辑三个方面进行多维度全面的安全审计。**经审计，FC-LiquidityProvider项目智能合约通过所有检测项，合约审计结果为通过。**以下为本合约详细审计信息。

代码规范审计

1. 编译器版本安全审计

老版本的编译器可能会导致各种已知的安全问题，建议开发者在代码中指定合约代码采用最新的编译器版本，并消除编译器告警。

- **安全建议：无**
- **审计结果：通过**

2. 弃用项审计

Solidity智能合约开发语言处于快速迭代中，部分关键字已被新版本的编译器弃用，如throw、years等，为了消除其可能导致的隐患，合约开发者不应该使用当前编译器版本已弃用的关键字。

- **安全建议：无**
- **审计结果：通过**

3. 冗余代码审计

智能合约中的冗余代码会降低代码可读性，并可能需要消耗更多的gas用于合约部署，建议消除冗余代码。

- **安全建议：无**
- **审计结果：通过**

4. require/assert 使用审计

Solidity使用状态恢复异常来处理错误。这种机制将会撤消对当前调用(及其所有子调用)中的状态所做的所有更改，并向调用者标记错误。函数assert和require可用于检查条件并在条件不满足时抛出异常。assert函数只能用于测试内部错误，并检查非变量。require函数用于确认条件有效性，例如输入变量，或合约状态变量是否满足条件，或验证外部合约调用的返回值。

- **安全建议：无**
- **审计结果：通过**

5. gas 消耗审计

波场虚拟机执行合约代码需要消耗gas，当gas不足时，代码执行会抛出out of gas异常，并撤销所有状态变更。合约开发者需要控制代码的gas消耗，避免因为gas不足导致函数执行一直失败。

- **安全建议：无**
- **审计结果：通过**

通用漏洞审计

1. 整型溢出审计

整型溢出是很多语言都存在的安全问题，它们在智能合约中尤其危险。Solidity最多能处理256位的数字(2**256-1)，最大数字增加1会溢出得到0。同样，当数字为uint类型时，0减去1会下溢得到最大数字值。溢出情况会导致不正确的结果，特别是如果其可能的结果未被预期，可能会影响程序的可靠性和安全性。

- **安全建议：无**
- **审计结果：通过**

2. 重入攻击审计

重入漏洞是最典型的智能合约漏洞，该漏洞原因是Solidity中的call.value()函数在被用来发送TRX的时候会消耗它接收到的所有gas，当调用call.value()函数发送TRX的逻辑顺序存在错误时，就会存在重入攻击的风险。

- **安全建议：无**
- **审计结果：通过**

3. 伪随机数生成审计

智能合约中可能会使用到随机数，在solidity下常见的是用block区块信息作为随机因子生成，但是这样使用是不安全的，区块信息是可以被矿工控制或被攻击者在交易时获取到，这类随机数在一定程度上是可预测或可碰撞的，比较典型的例子就是fomo3d的airdrop随机数可以被碰撞。

- **安全建议：无**
- **审计结果：通过**

4. 交易顺序依赖审计

在波场的交易打包执行过程中，面对相同难度的交易时，矿工往往会选择gas费用高的优先打包，因此用户可以指定更高的gas费用，使自己的交易优先被打包执行。

- **安全建议：无**
- **审计结果：通过**

5. 拒绝服务攻击审计

拒绝服务攻击，即Denial of Service，可以使目标无法提供正常的服务。在波场智能合约中也会存在此类问题，由于智能合约的不可更改性，该类攻击可能使得合约永远无法恢复正常工作状态。导致智能合约拒绝服务的原因有很多种，包括在作为交易接收方时的恶意revert、代码设计缺陷导致gas耗尽等等。

- **安全建议：无**
- **审计结果：通过**

6. 函数调用权限审计

智能合约如果存在高权限功能，如：铸币、自毁、change owner等，需要对函数调用做权限限制，避免权限泄露导致的安全问题。

- **安全建议：无**
- **审计结果：通过**

7. call/delegatecall安全审计

Solidity中提供了call/delegatecall函数来进行函数调用，如果使用不当，会造成call注入漏洞，例如call的参数如果可控，则可以控制本合约进行越权操作或调用其他合约的危险函数。

- **安全建议：无**

- **审计结果：通过**

8. 返回值安全审计

在Solidity中存在transfer()、send()、call.value()等方法中，transfer转账失败交易会回滚，而send和call.value转账失败会return false，如果未对返回做正确判断，则可能会执行到未预期的逻辑;另外在TRC20 Token的transfer/transferFrom功能实现中，也要避免转账失败return false的情况，以免造成假充值漏洞。

- **安全建议：无**
- **审计结果：通过**

9. tx.origin使用安全审计

在波场智能合约的复杂调用中，tx.origin表示交易的初始创建者地址，如果使用tx.origin进行权限判断，可能会出现错误;另外，如果合约需要判断调用方是否为合约地址时则需要使用tx.origin，不能使用extcodesize。

- **安全建议：无**
- **审计结果：通过**

10. 重放攻击审计

重放攻击是指如果两份合约使用了相同的代码实现，并且身份鉴权在传参中，当用户在向一份合约中执行一笔交易，交易信息可以被复制并且向另一份合约重放执行该笔交易。

- **安全建议：无**
- **审计结果：通过**

11. 变量覆盖审计

波场存在着复杂的变量类型，例如结构体、动态数组等，如果使用不当，对其赋值后，可能导致覆盖已有状态变量的值，造成合约执行逻辑异常。

- **安全建议：无**
- **审计结果：通过**

业务审计

1. 增加流动性功能

- **业务描述：**合约实现了 addLiquidity 功能用于添加流动性，要求调用者为合约的 owner。输入的 tokenAmount 不得超过合约里 token 代币的余额。之后通过调用输入的 LP 代币地址的 addLiquidity 函数注入流动性。

```
20    function addLiquidity(address lpAddr, address tokenAddr, uint256 tokenAmount) public payable returns (bool) {
21        require(msg.sender == _owner, "sender is not owner");
22
23        IERC20 token = IERC20(tokenAddr);
24        uint256 tokenBalance = token.balanceOf(address(this));
25        require(tokenBalance >= tokenAmount, "token is not sufficient");
26        token.approve(lpAddr, tokenAmount);
27
28        IJustswapExchange exchange = IJustswapExchange(lpAddr);
29        exchange.addLiquidity.value(msg.value)(1, tokenAmount, now + 60);
30
31        return true;
32    }
```

图 1 addLiquidity 函数源码

- **相关函数：**addLiquidity
- **安全建议：**无
- **审计结果：**通过

2. **提取指定代币功能**

- **业务描述：**合约实现了 withdraw 功能用于提取本合约里所有的指定代币到指定地址，要求调用者为合约的 owner。如果输入的 cntr 地址为 0 地址，则会提取本合约里所有的 TRX；如果不为 0 地址，则提取指定的 TRC20 代币。

```
34    function withdraw(address cntr, address payable recipient) public returns (bool) {
35        require(msg.sender == _owner, "sender is not owner");
36
37        if (cntr == address(0)) {
38            uint256 balance = address(this).balance;
39            if (balance > 0) {
40                recipient.transfer(balance);
41                return true;
42            } else {
43                return false;
44            }
45        } else {
46            IERC20 token = IERC20(cntr);
47            uint256 balance = token.balanceOf(address(this));
48            if (balance > 0) {
49                token.transfer(recipient, balance);
50                return true;
51            } else {
52                return false;
53            }
54        }
55    }
56 }
```

图 2 withdraw 函数源码

- **相关函数：**withdraw
- **安全建议：**无
- **审计结果：**通过

结论

Beosin(成都链安)对 FC-LiquidityProvider 项目的智能合约的设计和代码实现进行了详细的审计。审计团队在审计过程中发现的问题均已告知项目方修复，FC-LiquidityProvider 项目的智能合约的总体审计结果是**通过。**

成都链安
BEOSIN

官方网址

https://lianantech.com

电子邮箱

vaas@lianantech.com

微信公众号