ParcelX 代码审计报告

Code Auditing Report for ParcelX

漏洞银行 | BUGBANK

版本记录

版本号	发布时间	修改内容和原因	作者	审核
1.0	2018/6/13	文档初稿完成	Faith	Bowen
1.1	2018/6/14	文档终稿完成	Faith	Bowen

目 录

一、 概述

ParcelX 是一个已在中国、韩国、日本建立了市场基础的物流网络。拥有多年的连接跨境电子商务递送服务供应商的经验，其中包括 30 家全球航空运载商和 20 个国家邮政系统平台。ParcelX 拥有中国港口的独家运营权，在日本和韩国也都设有海外仓库。

源代码审计工作通过分析当前智能合约的源代码，熟悉业务逻辑，从安全性方面检查其脆弱性和缺陷。在明确当前安全现状和需求的情况下，对下一步的编码安全规范性建设有重大的意义。

源代码审计工作利用一定的编程规范和标准，针对应用程序源代码，从结构、脆弱性以及缺陷等方面进行审查，以发现当前应用程序中存在的安全缺陷以及代码的规范性缺陷。。

1.1 项目目标

本次源代码审计工作是通过对当前智能合约的源代码进行审查，以检查代码在程序编写上可能引起的安全性和脆弱性问题。

1.2 项目范围

经我司安全分析部门同 ParcelX 相关负责部门确认，对客户提供需要审计的源代码进行代码审计，代码地址如下：

NO.	代码地址
1	https://github.com/ParcelX/tokensale/tree/194ea1c67912fb7ffef6f9060890e5e40b74871d

1.3 项目内容

通过代码审计的方式检查智能合约的安全性，白盒测试所采用的方法是工具审查结合人工审查，依照 DASP TOP 10 所披露的脆弱性，结合智能合约开发规范，检查目标系统的脆弱性、缺陷以及结构上的问题。

二、 代码审计综述

2.1 漏洞概述

漏洞名称	漏洞描述	风险等级	是否存在
任意转移token	允许从任意用户的balances中进行转移token	高	否
整型上溢	对数据进行数学计算时产生整型上溢漏洞	高	否
整型下溢	对数据进行数学计算时产生整型下溢漏洞	高	否
短地址攻击	短地址补齐导致转账数量异常	高	是
重入漏洞	调用恶意合约时，可被恶意合约操作	中	是
访问控制	对关键操作的访问无限制导致合约可被恶意修改控制	中	否
底层函数调用检查	底层函数调用返回结果判断	中	是
时间操纵	对依赖矿工提供的时间的恶意操纵	中	否
硬编码地址	硬编码钱包地址在合约中	低	否
编译版本不固定	编译版本不固定导致兼容性和稳定性问题	低	是
复杂fallback函数	Fallback复杂导致send()和	低	是

	transfer()转账给合约时因gas不足失败		
条件竞争	**不同的操作存在先后顺序，不同顺序可导致不同结果**	**低**	**是**
优先运行漏洞	**增加gas可使自己的交易优先被处理从而获利**	**低**	**否**
FOR循环拒绝服务	**For循环变量类型不匹配导致循环无法结束拒绝服务**	**低**	**否**
随机数安全问题	**随机数可预测漏洞**	**低**	**否**

三、 审计详情

3.1 编译版本不固定

3.1.1 内容描述

编译版本不固定，可造成兼容问题和稳定问题。

3.1.2 合约审计

合约中使用代码如下

```
pragma solidity ^0.4.19;
```

需要一定范围内的编译器才能正常编译。

3.1.3 审计结论

推荐使用固定的编译器版本，保证代码的稳定性。

```
pragma solidity 0.4.19;
```

3.2 复杂 fallback 函数

3.2.1 内容描述

Fallback 函数涉及到其他函数操作，需要较多的 gas，当使用.send()或.transfer()进行转账时，只有 2300 gas，无法完成复杂操作，导致失败。

3.2.2 合约审计

fallback 函数源码

```
 function () payable public {
        if (msg.value > 0) {
            buy();
```



```
        }
    }
```

3.2.3 审计结论

合约 fallback 函数会调用 buy 函数，需要额外 gas，会导致使用 send 或 transfer 转账失败。

3.3 任意转移 token

3.3.1 内容描述：

transferFrom 函数中，如果未对 allowed[_from][msg.sender]和_value 的大小进行判断，则存在从任意账号中获取 token 的漏洞。

3.3.2 合约审计：

```
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);

        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
```

```
        Transfer(_from, _to, _value);
        return true;
    }
```

3.3.3 审计结论：

合约中存在 require(_value <= allowed[_from][msg.sender]);判断。

无此漏洞。

3.4 重入漏洞

3.4.1 内容描述

当合约调用其他恶意合约时，如进行转账操作，使用 call 函数进行转账时，会提供所有剩余的 gas，当接收方为恶意的智能合约时，其 fallback 函数可对该智能合约进行额外的操作，导致漏洞。

3.4.2 合约审计

```
    function execute(address _to, uint256 _value, bytes _data) mostOwner(keccak256(msg.data)) external returns (bool){
        require(_to != address(0));
        Withdraw(_to, _value, msg.sender);
        return _to.call.value(_value)(_data);
    }
```

3.4.3 审计结论

建议使用.transfer()函数进行转账，只传递 2300gas，可有效防止重入漏洞。

3.5 访问控制

3.5.1 内容描述

关键函数未进行访问控制，可被任意调用，则存在风险。

3.5.2 合约审计

setBuyRate()和 execute()函数均使用修饰符 mostOwner 进行判断，对其操作需要多人同意。

3.5.3 审计结论

无此漏洞

3.6 条件竞争

3.6.1 内容描述：

ERC20 已知漏洞。https://github.com/ethereum/EIPs/issues/738

ERC20 中存在 2 个函数，approve(address _spender, uint256 _value) 和 transferFrom(address _from, address _to, uint256 _value)，当条件竞争时，存在多花的漏洞。

利用场景：

1.A 调用 approve(B,100)

2.A 改变主意，调用 approve(B,20)

3.在 approve(B,10)还没有被矿工挖处之前，B 调用 transFrom(A,B,100)

4.如果 transFrom(A,B,100)在 approve(B,20)之前被挖出，则 B 还可以调用 transFrom(A,B,20)，最终转账 120 个 token.

3.6.2 合约审计

合约中实现了 increaseApproval()和 decreaseApproval()函数对 allowed 进行操作，可部分避免上述问题，需要用户对 approval 进行操作时，调用上述函数进行。

decreaseApproval 函数中，当用户_subtractedValue 大于 oldValue 时，对 allowed[msg.sender][_spender]进行了赋值 0 的操作，与函数本意存在一定歧义，用户使用时，需要明白自己的操作。

3.6.3 审计结论

用户使用时，

1）对同一_spender 多次使用 approve 函数

2）使用 decreaseApproval 函数

需要明确实际操作可能存在的风险。

3.7 整型上溢

3.7.1 内容描述：

对整型数据进行加法/乘法数学运算时，如未检测操作结果是否在范围内，则存在整型上溢的问题。

3.7.2 合约审计

ParcelXGPX.sol 文件中对数据的加法/乘法均使用 SafeMath 函数进行操作，安全。

3.7.3 审计结论

无漏洞。

3.8 整型下溢

3.8.1 内容描述：

对整型数据进行减法数学运算时，如未检测操作结果是否在范围内，则存在整型下溢的问题。

3.8.2 合约审计

```
function decreaseApproval(address _spender, uint _subtractedValue) public returns (bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
```

合约中减法操作使用 SafeMath 中的 sub 函数进行，同时也存在大小的判断 _subtractedValue > oldValue。

3.8.3 审计结论

无漏洞

3.9 底层函数调用检查

3.9.1 内容描述

当使用 call, callcode, delegatecall, send 操作时，如果未检测返回值，存在操作失败的风险。

3.9.2 合约审计

合约中有一处 call 调用，

```
function execute(address _to, uint256 _value, bytes _data) mostOwner(keccak256(msg.data)) external returns (bool){
        require(_to != address(0));
        Withdraw(_to, _value, msg.sender);
        return _to.call.value(_value)(_data);
    }
```

3.9.3 审计结论

对函数返回结果进行检查。

3.10 FOR 循环拒绝服务

3.10.1 内容描述：

for (var i = 0; i < arrayName.length; i++) { ... } i 为 uint8，如果 arrayName.length 大于 255，则循环操作无法退出。

3.10.2 合约审计

MultiOwnable.sol 中循环变量 i 均为 uint 类型，且与判断变量为同一类型。

3.10.3 审计结论

无此漏洞

3.11 随机数安全问题

3.11.1 内容描述：

合约使用不安全的随机数，存在被控制的风险。

3.11.2 合约审计

本合约中未发现对随机数的调用。

3.11.3 审计结论

无此漏洞

3.12 优先运行漏洞

3.12.1 内容描述：

恶意用户可以支付更高费用来让交易操作得到优先收录。

3.12.2 合约审计

合约中不存在优先提交答案获利的情形。

3.12.3 审计结论

无影响。

3.13 时间操纵

3.13.1 内容描述：

矿工可以对 now 函数进行控制，依赖 now 函数返回值的判断存在错误的风险。

3.13.2 合约审计

本合约中未涉及到对时间的获取。

3.13.3 审计结论

无此漏洞

3.14 短地址攻击

3.14.1 内容描述

transfer(address _to, uint256 _amount) 当用户的地址尾部存在 0 时，如果用户提供给 transfer 函数地址尾部 0 被去掉，则以太坊虚拟机执行转账操作时，_amount 会补充到_to 中，最终_amount 会后端补 0，造成转出更多 token.

3.14.2 合约审计

合约中重写的 ERC20 标准中的 transfer 函数未增加参数检测，可能会受到短地址攻击：

```
function transfer(address _to, uint256 _value) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[msg.sender]);

        // SafeMath.sub will throw if there is not enough balance.
        balances[msg.sender] = balances[msg.sender].sub(_value);
```



```
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);
        return true;
    }
```

3.14.3 审计结论

修复方法：

```
    modifier onlyPayloadSize(uint size) {
        assert(msg.data.length == size + 4);
        _;
      }

    function transfer(address _to, uint256 _value) onlyPayloadSize(2 * 32) public
returns (bool) {
            require(_to != address(0));
            require(_value <= balances[msg.sender]);

            // SafeMath.sub will throw if there is not enough balance.
            balances[msg.sender] = balances[msg.sender].sub(_value);
            balances[_to] = balances[_to].add(_value);
            Transfer(msg.sender, _to, _value);
            return true;
```


modifier 中的 size 根据参数数量进行调整，当前应采用 2 * 32。

3.15 硬编码地址

3.15.1 内容描述

使用硬编码地址，当丢失此地址的访问权限时，存在币丢失的风险。

3.15.2 合约审计

合约中未发现使用硬编码地址

3.15.3 审计结论

无此漏洞

