# Smart Contract Security Audit Report


# Revision History

| Version | Date | Changes | Author | Auditor |
|---|---|---|---|---|
| 1.0 | 2018/6/20 | First draft created | Eric | Bowen |
| 1.1 | 2018/6/22 | Final report finished | Eric | Bowen |
| | | | | |
| | | | | |
| | | | | |

# Table of Contents

# 1. Introduction

ParcelX is a logistics network with an established market base in China, Korea and Japan. It possesses years of experience connecting cross-border e-commerce delivery providers, including 30 global air carriers and 20 national postal system platforms. ParcelX has exclusive ownership of ports China, with overseas warehouses at Japan and Korea.

We reviewed and audited the tokensale smart contract of ParcelX.

## 1.1 Goal

The audit for the source code aims at auditing all modules of the smart contract and identify any security problems.

## 1.2 Scope

Confirmed with ParcelX, the scope of code audit is the project tokensale source code on Github:

| NO. | Source code |
|---|---|
| 1 | https://github.com/ParcelX/tokensale/tree/194ea1c67912fb7ffef6f9060890e5e40b74871d |

The standard of smart contract audit is based on the DASP TOP 10 and rules of smart-contract-best-practices.

 

# 2. Main Results

## 2.1 Vulnerability types

We reviewed the smart contract and audited 15 types of vulnerabilities. The severity includes three levels: high, medium and low.

| Vulnerability | Description | Severity | Exist |
|---|---|---|---|
| Transfer Token Arbitrarily | Attackers can take token from arbitrary address if the values of allowed[_from][msg.sender] and _value in function transferFrom are not checked. | High | No |
| Integer Overflow | If a balance reaches the maximum uint value (2^256) it will circle back to zero. | High | No |
| Integer Underflow | If a uint is made to be less than zero, it will cause an underflow and get set to its maximum value. | High | No |
| Short Addresses | Short address attacks are a side-effect of the EVM itself accepting incorrectly padded arguments. | High | Yes |
| Reentrancy | Reentrancy occurs when external contract calls are allowed to make new calls to the calling contract before the initial execution is complete. | Medium | Yes |

 

| | | | |
|---|---|---|---|
| Access Control | The critical function can be called arbitrarily. | Medium | No |
| Unchecked Low Level Calls | If the return value of low-level calls are not checked, can lead to fail-opens and other unwanted outcomes. | Medium | Yes |
| Time Manipulation | A transaction's miner has leeway in reporting the time at which the mining occurred, good smart contracts will avoid relying strongly on the time advertised. | Medium | No |
| Hardcoded Addresses | When using Hardcoded addresses, the property will be lost if the addresses are not accessible. | Low | No |
| Changeable Compiler Version | Compilers with different versions may have unsupported functions between each other | Low | Yes |
| Complex Fallback Function | The fallback function can only rely on 2300 gas being available (for example when send or transfer is used), leaving little room to perform other operations except basic logging. | Low | Yes |

 

| | | | |
|---|---|---|---|
| Race Conditions | Attackers can take over the control flow, and make changes to your data that the calling function wasn't expecting. | Low | Yes |
| Front Running | Attackers can specify higher fees to have their transactions mined more quickly. | Low | No |
| Denial of Services | Many ways lead to denials of service, including maliciously behaving when being the recipient of a transaction, artificially increasing the gas necessary to compute a function, etc. | Low | No |
| Bad Randomness | Sources of randomness are to an extent predictable, malicious users can generally replicate it and attack the function relying on its unpredictability. | Low | No |

 

# 3. Findings

## 3.1 Changeable Compiler Version

### 3.1.1 Description

Compilers with different versions may have unsupported functions between each other.

### 3.1.2 Detail

```
pragma solidity ^0.4.19;
```

The contracts can be compiled by a series of compiler with different versions.

### 3.1.3 Proposal

Using compiler with specific version to ensure the stability of codes.

```
pragma solidity 0.4.19;
```

Vulnerable contracts: Buyable.sol, ERC20.sol, MultiOwnable.sol, Pausable.sol, Sample.sol, Convertible.sol, Migrations.sol, ParcelXGPX.sol, SafeMath.sol

## 3.2 Complex Fallback Function

### 3.2.1 Description

The fallback function can only rely on 2300 gas being available (for example when .send() or .transfer() is used), leaving little room to perform other operations except basic logging.

 

3.2.2 Detail

The source code of fallback function:

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

The fallback function will call buy(), which will consume extra gas and cause the failure of transferring with send() or transfer().

Vulnerable contracts: ParcelXGPX.sol

## 3.3 Transfer Token Arbitrarily

3.3.1 Description

Attackers can take tokens from an arbitrary address if the values of allowed[_from][msg.sender] and _value in function transferFrom are not checked.

3.3.2 Detail：

```
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 Proposal：

Contracts don't require changes.

## 3.4 Reentrancy

3.4.1 Description

Reentrancy occurs when external contract calls are allowed to make new calls to the calling contract before the initial execution is complete.

3.4.2 Detail

```
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 Proposal

Transfer ethers with .transfer()，which consumes 2300 gas and can avoid reentrancy vulnerability。

Vulnerable contracts: ParcelXGPX.sol

## 3.5 Access Control

3.5.1 Description

The critical function can be called arbitrarily.

3.5.2 Detail

setBuyRate() and execute() both use decorator mostOwner to check actions, which is safe enough.

3.5.3 Proposal

Contracts don't require changes.

## 3.6 ERC20 Race Conditions

3.6.1 Description

Attackers can take over the control flow, and make changes to your data that the calling function wasn't expecting.

 

3.6.2 Detail：

Vulnerability of ERC20: https://github.com/ethereum/EIPs/issues/738

These two functions in ERC20: approve(address _spender, uint256 _value) and transferFrom(address _from, address _to, uint256 _value) will cause multiple spend when the race conditions vulnerability exists. The function increaseApproval() and decreaseApproval() in contract can avoid this problem.

3.6.3 Proposal

Contracts don't require changes.

## 3.7 Integer Overflow

3.7.1 Description

If a balance reaches the maximum uint value (2^256) it will circle back to zero.

3.7.2 Detail

The critical data is manipulated with SafeMath, it's safe enough.

3.7.3 Proposal

Contracts don't require changes.

## 3.8 Integer Underflow

3.8.1 Description

If a uint is made to be less than zero, it will cause an underflow and get set to its maximum value.

 

3.8.2 Detail

```
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;
    }
```

The critical data is manipulated with SafeMath and the contract has values comparison: _subtractedValue > oldValue, so it's safe enough.

3.8.3 Proposal

Contracts don't require changes.

## 3.9 Unchecked Low Level Calls

3.9.1 Description

If the return value of low-level calls are not checked, can lead to fail-opens and other unwanted outcomes.

 

3.9.2 Detail

The .call.value() is called in the contract.

```
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 Proposal

Transfer ethers with .transfer().

Vulnerable contracts: ParcelXGPX.sol

## 3.10 Denial of Services

3.10.1 Description

Many ways lead to denials of service, including maliciously behaving when being the recipient of a transaction, artificially increasing the gas necessary to compute a function, abusing access controls to access private components of smart contracts, taking advantage of errors and negligence, loop will not terminate because of type of arguments, etc.

 

3.10.2 Detail

The type of argument i is uint in contract MultiOwnable.sol, and is the same as the type of _multiOwners.

3.10.3 Proposal

Contracts don't require changes.

## 3.11 Bad Randomness

3.11.1 Description

Sources of randomness are to an extent predictable, malicious users can generally replicate it and attack the function relying on its unpredictability.

3.11.2 Detail

Randomness doesn't exist in the contracts.

3.11.3 Proposal

Contracts don't require changes.

## 3.12 Front Running

3.12.1 Description

Attackers can specify higher fees to have their transactions mined more quickly.

3.12.2 Detail

Function transferFrom() and approve() have this problem.

 

3.12.3 Proposal

Executing the functions with obeying the rule.

Vulnerable contracts: ParcelXGPX.sol

## 3.13 Time Manipulation

3.13.1 Description

A transaction's miner has leeway in reporting the time at which the mining occurred and good smart contracts will avoid relying strongly on the time advertised.

3.13.2 Detail

These contracts don't get time, so they don't have this vulnerability.

3.13.3 Proposal

Contracts don't require changes

## 3.14 Short Addresses

3.14.1 Description

Short address attacks are a side-effect of the EVM itself accepting incorrectly padded arguments.

3.14.2 Detail

The transfer() function doesn't check the bytes of arguments, it may be vulnerable to the short addresses attack.

 

```
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;
```

Proposal

```
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;
```

The argument "size" in onlyPayloadSize decorator depends on the number of arguments in function transfer()

Vulnerable contracts: ParcelXGPX.sol

## 3.15 Hardcoded Addresses

### 3.15.1 Description

When using Hardcoded addresses, the property will lose if the addresses are not accessible.

### 3.15.2 Detail

Hardcoded address is not found in these contracts.

### 3.15.3 Proposal

Contracts don't require changes

 