密级 ★ | 项目组公开

[文档编号：LXT-DMSJ-20180709]

智能合约审计报告

LXT

主测人：

知道创宇

版本说明

修订人	修订内容	修订时间	版本号	审阅人
于超	编写文档	**2018/7/9**	V1.0	于超

文档信息

文档名称	**文档版本号**	文档编号	**保密级别**
LXT 智能合约审计报告	V1.0	【LXT-DMSJ-20180709】	项目组公开

版权声明

目录

1. 综述

本次报告有效测试时间是从 2018 年 7 月 9 日开始到 2018 年 7 月 9 日结束，在此期间针对 **LXT 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，发现 LXT 合约代码存在事务顺序依赖风险，该风险较难利用，故综合评定为安全。

本次智能合约安全审计结果： 通过

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次测试的目标信息：

模块名称	
Token 名称	LiteXToken
合约类型	代币合约
合约地址	0xBC46D9961A3932f7D6b64abfdeC80C1816C4B835
代码链接	https://etherscan.io/address/0xbc46d9961a3932f7d6b64abfdec80c1816c4b835#code
代码语言	Solidity

本次项目测试人员信息：

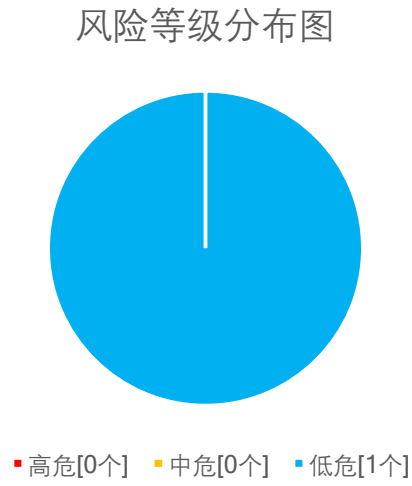
姓名	邮箱/联系方式	职务
于超	yuc@knownsec.com	高级工程师

2. 代码漏洞分析

2.1. 漏洞等级分布

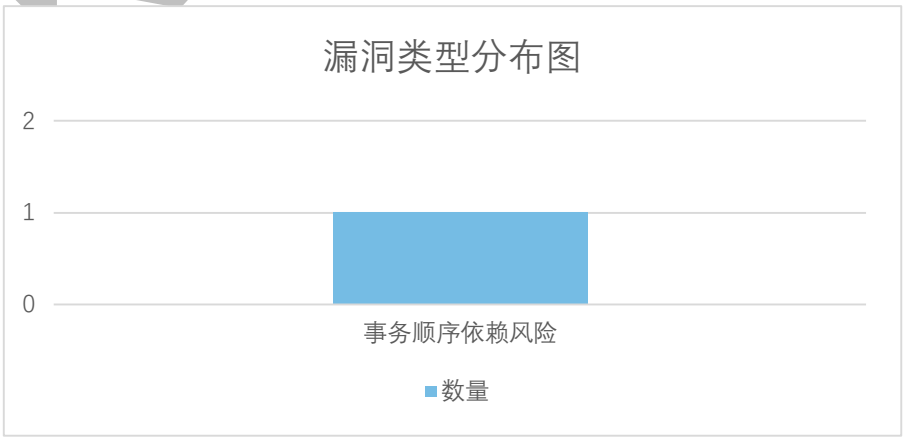
本次漏洞风险按等级统计：

漏洞风险等级个数统计表		
高危	中危	低危
0	0	1



2.2. 漏洞类型分布

本次漏洞按类型统计：



2.3. 审计结果汇总

审计结果			
测试项目	测试内容	状态	描述
智能合约	**重入攻击检测**	安全	检查 call.value()函数使用安全
	数值溢出检测	安全	检查 add 和 sub 函数使用安全
	访问控制缺陷检测	安全	检查各操作访问权限控制
	未验证返回值的调用	安全	检查转币方法看是否验证返回值
	错误使用随机数检测	安全	检查是否具备统一的内容过滤器
	事务顺序依赖检测	低危	检查事务顺序依赖
	拒绝服务攻击检测	安全	检查代码在使用资源时是否存在资源滥用问题
	逻辑设计缺陷检测	安全	检查智能合约代码中与业务设计相关的安全问题

3. 代码审计结果分析

3.1. 重入攻击检测【安全】

重入漏洞是最著名的以太坊智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在相关 call 外部合约调用。

安全建议：无。

3.2. 数值溢出检测【安全】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 为的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.3. 访问控制检测【安全】

访问控制缺陷是所有程序中都可能存在的安全风险，智能合约也同样会存在类似问题，著名的 Parity Wallet 智能合约就受到过该问题的影响。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无。

3.4. 返回值调用验证【安全】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无。

3.5. 错误使用随机数【安全】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问

明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数载一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该问题。

安全建议：无。

3.6. 事务顺序依赖【低危】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于以太坊区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，LXT 代币合约中的 approve 函数中存在事务顺序依赖攻击风险，代码如下：

```
157  /**
158   * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
159   *
160   * Beware that changing an allowance with this method brings the risk that someone may use both the old
161   * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
162   * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
163   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
164   * @param _spender The address which will spend the funds.
165   * @param _value The amount of tokens to be spent.
166   */
167  function approve(address _spender, uint256 _value) public returns (bool) {
168    allowed[msg.sender][_spender] = _value;
169    Approval(msg.sender, _spender, _value);
170    return true;
171  }
```

可能存在的安全风险描述如下：

1. 用户 A 通过调用 approve 函数允许用户 B 代其转账的数量为 N（N>0）；

2. 经过一段时间后，用户 A 决定将 N 改为 M（M>0），所以再次调用 approve 函

数；

3. 用户 B 在第二次调用被矿工处理之前迅速调用 transferFrom 函数转账 N 数量的 token；

4. 用户 A 对 approve 的第二次调用成功后，用户 B 便可再次获得 M 的转账额度，即用户 B 通过交易顺序攻击获得了 N+M 的转账额度。

安全建议：

1. 前端限制，当用户 A 将额度从 N 修改为 M 时，可先从 N 修改为 0，再从 0 修改为 M。

2. 在 approve 函数开头增加如下代码：

```
require((_value == 0) || (allowed[msg.sender][_spender] == 0));
```

3.7. 拒绝服务攻击【安全】

在以太坊的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能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可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无。

3.8. 逻辑设计缺陷【安全】

检测智能合约代码中与业务设计相关的安全问题。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无。

4. 附录 A：合约代码

https://etherscan.io/address/0xbc46d9961a3932f7d6b64abfdec80c1816c4b835#code

```
pragma solidity ^0.4.18;

// File: zeppelin-solidity/contracts/math/SafeMath.sol

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  /**
  * @dev Substracts two numbers, throws on overflow (i.e. if subtrahend is greater
than minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}

// File: zeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

// File: zeppelin-solidity/contracts/token/ERC20/BasicToken.sol
```

```
/**
* @title Basic token
* @dev Basic version of StandardToken, with no allowances.
*/
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances;

  uint256 totalSupply_;

  /**
  * @dev total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  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;
  }

  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  function balanceOf(address _owner) public view returns (uint256 balance) {
    return balances[_owner];
  }

}

// File: zeppelin-solidity/contracts/token/ERC20/ERC20.sol

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns
(bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: zeppelin-solidity/contracts/token/ERC20/StandardToken.sol

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * @dev https://github.com/ethereum/EIPs/issues/20
 * @dev Based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;

  //knownsec//使用了SafeMath防止数值溢出
  /**
```

```
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amount of tokens to be transferred
   */
  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;
  }
  //knownsec//approve函数存在事务顺序依赖风险，详见3.6章节
  /**
   * @dev Approve the passed address to spend the specified amount of tokens on behalf
of msg.sender.
   *
   * Beware that changing an allowance with this method brings the risk that someone
may use both the old
   * and the new allowance by unfortunate transaction ordering. One possible solution
to mitigate this
   * race condition is to first reduce the spender's allowance to 0 and set the
desired value afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   * @param _spender The address which will spend the funds.
   * @param _value The amount of tokens to be spent.
   */
  function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
  }

  /**
   * @dev Function to check the amount of tokens that an owner allowed to a spender.
   * @param _owner address The address which owns the funds.
   * @param _spender address The address which will spend the funds.
   * @return A uint256 specifying the amount of tokens still available for the
spender.
   */
  function allowance(address _owner, address _spender) public view returns (uint256) {
    return allowed[_owner][_spender];
  }

  /**
   * @dev Increase the amount of tokens that an owner allowed to a spender.
   *
   * approve should be called when allowed[_spender] == 0. To increment
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _addedValue The amount of tokens to increase the allowance by.
   */
  function increaseApproval(address _spender, uint _addedValue) public returns (bool)
{
    allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
    Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

  /**
   * @dev Decrease the amount of tokens that an owner allowed to a spender.
   *
   * approve should be called when allowed[_spender] == 0. To decrement
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _subtractedValue The amount of tokens to decrease the allowance by.
   */
```

```
  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;
  }

}

// File: contracts/LiteXToken.sol

contract LiteXToken is StandardToken {

  string public name = "LiteXToken";
  string public symbol = "LXT";
  uint8 public decimals = 18;
  uint INITIAL_SUPPLY = 20*10**8; // 2 billion tokens

  function LiteXToken() public {
    totalSupply_ = INITIAL_SUPPLY*10**uint256(decimals);
    balances[msg.sender] = totalSupply_;
  }

  function() public payable{
    revert();
  }


}
```

5. 附录 B：漏洞测试工具简介

5.1. Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号以太坊虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。 与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

5.2. Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合

约中自定义的属性。

5.3. securify.sh

Securify 可以验证以太坊智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

5.4. Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

5.5. MAIAN

MAIAN 是一个用于查找以太坊智能合约漏洞的自动化工具，Maian 处理合约的字节码，并尝试建立一系列交易以找出并确认错误。

5.6. ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

5.7. ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

5.8. Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建以太坊合约并调试交易。

5.9. 知道创宇渗透测试人员专用工具包

知道创宇渗透测试人员专用工具包，由知道创宇渗透测试工程师研发，收

集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

[咨询电话] +86(10)400 060 9587
[投诉电话] 13811527185
[邮　　箱] sec@knownsec.com
[网　　址] www.knownsec.com
[地　　址] 北京市朝阳区望京SOHO T3 A座15层

北京知道创宇信息技术有限公司