密级 ★ | 项目组公开

[文档编号：SEC-DMSJ-20180622]

智能合约审计报告

SEC

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
代其	编写文档	**2018/6/22**	V1.0	

文档信息

文档名称	**文档版本**	文档编号	**保密级别**
SEC 智能合约审计报告	V1.0	【SEC-DMSJ-20180622】	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 6 月 20 日开始到 2018 年 6 月 22 日结束，在此期间针对 SEC **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，未发现目标代码存在合约安全漏洞，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	SEC
代码类型	代币代码
合约地址	*0xc6689eb9a6d724b8d7b1d923ffd65b7005da1b62*
链接地址	*https://etherscan.io/address/0xc6689eb9a6d724b8d7b1d923ffd65b7005da1b62#code*
代码语言	solidity

本次项目测试人员信息：

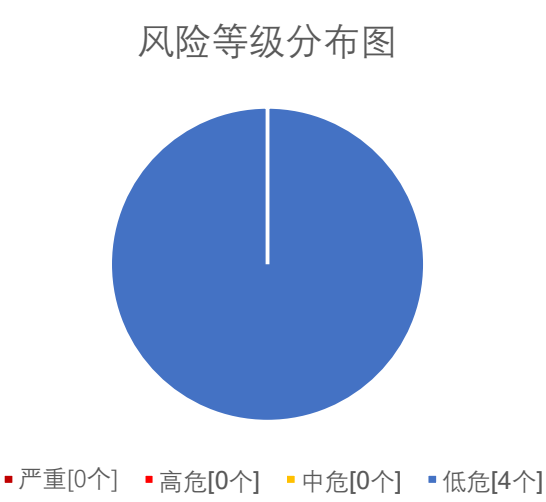
姓名	邮箱/联系方式	职务
代其	daiq@knownsec.com	高级工程师
李鑫	lix2@knownsec.com	总工

2. 代码漏洞分析

2.1. 漏洞等级分布

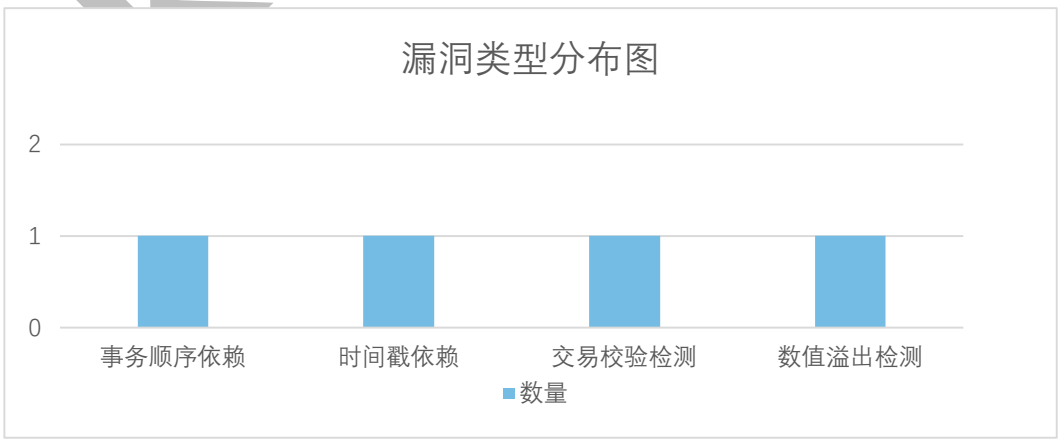
本次漏洞风险按等级统计：

漏洞风险等级个数统计表			
严重	高危	中危	低危
0	0	0	4



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	低危	检查 add 和 sub 函数使用安全
	重放攻击检测	安全	检查操作是否可以重放
	重排攻击检测	安全	检查列表和映射操作，看是否存在信息插入问题
	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
	时间戳依赖检测	低危	检查时间戳使用是否存在安全隐患
	短地址攻击检测	安全	检查是否存在短地址攻击
	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
	交易校验检测	低危	检测交易前后买卖数据转换是否发生变化

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，智能合约代码中不涉及相关 call 外部合约调用。

安全建议：无。

3.2. 数值溢出检测【低危】

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，智能合约代码中使用存在该安全问题。管理员调用铸币函数给某个地址增加 token 时，利用溢出漏洞可以突破该函数只能增加 token 的限制，实际减少该地址的 token 数量，从而实现对任一账户余额的任意篡改（增加或减少）。在该合约中同时会使 totalSupply 同时溢出，利用代价太大，因此风

险性较小。

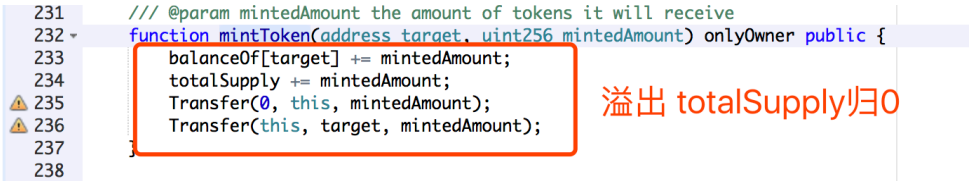


图 3.2 存在溢出问题

安全建议：在函数开头添加

require((balanceOf[target] + mintedAmount)> balanceOf[target]);

require((balanceOf[target] + mintedAmount)> balanceOf[target]);

3.3. 重放攻击检测【不涉及】

重放攻击是指在像以太坊这样的区块链上发起一笔交易，而后在像以太坊经典这样的另一个链上重复这笔交易的攻击。

检测结果：经检测，智能合约代码中不涉及相关漏洞。

安全建议：无。

3.4. 重排攻击检测【不涉及】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表（list）或映射（mapping）中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果：经检测，智能合约代码中不存在参与“竞争”相关漏洞。

安全建议：无。

3.5. 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，智能合约代码中敏感操作不存在相关漏洞。

```
/// @param mintedAmount the amount of tokens it will receive
function mintToken(address target, uint256 mintedAmount) onlyOwner public {
    balanceOf[target] += mintedAmount;
    totalSupply += mintedAmount;
    Transfer(0, this, mintedAmount);
    Transfer(this, target, mintedAmount);
}

/// @notice `freeze? Prevent | Allow` `target` from sending & receiving tokens
/// @param target Address to be frozen
/// @param freeze either to freeze it or not
function freezeAccount(address target, bool freeze) onlyOwner public {
    frozenAccount[target] = freeze;
    FrozenFunds(target, freeze);
}
```

图 3.5 身份访问控制

安全建议：无。

3.6. 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，智能合约代码中交易均使用 transfer，不存在相关漏洞。

```
211         if(teamaccount == sender){
212             if (now < start + 365 * 1 days) {
213                 require((balance - amount) >= SECtotalAmount/10 * 3/4);
214
215             } else if (now < start + (2*365+1) * 1 days){
216                 require((balance - amount) >= SECtotalAmount/10 * 2/4);
217
218             }else if (now < start + (3*365+1) * 1 days){
219                 require((balance - amount) >= SECtotalAmount/10 * 1/4);
220
221             }
222         }
223
224         balanceOf[_from] -= _value;                         // Subtract from the sender
225         balanceOf[_to] += _value;                           // Add the same to the recipient
226         Transfer(_from, _to, _value);
227     }
```

图 3.6 transfer 交易

安全建议：无。

3.7. 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，智能合约代码中不涉及该问题。

安全建议：无。

3.8. 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，该合约中存在事物顺序依赖攻击风险，使用了以太坊官方文档的示例代码 approve，而示例代码中存在事物顺序依赖攻击风险，代码如下

```
112      */
113     function approve(address _spender, uint256 _value) public
114     returns (bool success) {
115         allowance[msg.sender][_spender] = _value;
116         return true;
117     }
118
```

图 3.8 事务顺序依赖

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) || (allowance[msg.sender][_spender] == 0));

3.9. 时间戳依赖【低危】

如果智能合约中有在特定时间为游戏解锁资金等业务逻辑，则需要依赖当前时间，在 Solidity 中可通过 block.timestamp 或 now 还获取，但该时间戳在一定程度上可以被矿工控制。矿工可以通过为他正在挖掘的矿区选择合适的时间戳来获得优势。由于矿工可以影响他们区块的时间戳（在一定程度上），他们可以尝试挖掘一个包含他们交易的区块，并在未来设定一个区块时间戳。如果足够接近，它将在网络上被接受。

检测结果：经检测，该锁仓合约中调用了 now 来进行时间节点判断。

```
200
201     //about lock coins
202     uint start = 1520956799;
203     uint256 SECtotalAmount = 1500000000 * 10 ** 18;
204     address teamaccount = 0xC32b1519A0d4E883FE136AbB3198cbC402b5503F;
205
206     uint256 amount = _value;
207     address sender = _from;
208     uint256 balance = balanceOf[_from];
209
210
211     if(teamaccount == sender){
212         if (now < start + 365 * 1 days) {
213             require((balance - amount) >= SECtotalAmount/10 * 3/4);
214
215         } else if (now < start + (2*365+1) * 1 days){
216             require((balance - amount) >= SECtotalAmount/10 * 2/4);
217
218         }else if (now < start + (3*365+1) * 1 days){
219             require((balance - amount) >= SECtotalAmount/10 * 1/4);
220
221         }
222     }
223
224     balanceOf[_from] -= _value;                         // Subtract from the sender
225     balanceOf[_to] += _value;                           // Add the same to the recipient
226     Transfer(_from, _to, _value);
227 }
```

图 3.9 now 进行时间戳判断

安全建议：建议将现有的时间戳概念转换为块编号。

3.10. 短地址攻击【安全】

短地址攻击是 EVM 本身接受不正确填充参数的副作用。攻击者可以通过使制作的地址来利用这一点，使编码错误的客户端在将它们包含在事务之前不正确地对参数进行编码。从而实现诱骗参与者将 ethers 发送到攻击者的地址。

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。

3.11. 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。

3.12. 交易校验检测【低危】

在智能合约中，在交易前后买卖双方的数据转换需要校验，防止因为漏洞导致数据变更造成经济损失。

检测结果：经检测该智能合约中，缺少 uint previousBalances = balanceOf[_from] + balanceOf[_to];的相关校验

```
194   function _transfer(address _from, address _to, uint _value) internal {
195       require (_to != 0x0);                               // Prevent transfer to 0x0 address. Use burn() ir
196       require (balanceOf[_from] >= _value);               // Check if the sender has enough
197       require (balanceOf[_to] + _value > balanceOf[_to]); // Check for overflows
198       require(!frozenAccount[_from]);                     // Check if sender is frozen
199       require(!frozenAccount[_to]);                       // Check if recipient is frozen
200
201       //about lock coins
202       uint start = 1520956799;
203       uint256 SECtotalAmount = 1500000000 * 10 ** 18;
204       address teamaccount = 0xC32b1519A0d4E883FE136AbB3198cbC402b5503F;
205
206       uint256 amount = _value;
207       address sender = _from;
208       uint256 balance = balanceOf[_from];        缺少数值运行校验
209
210   uint previousBalances = balanceOf[_from] + balanceOf[_to];
211       if(teamaccount == sender){
212           if (now < start + 365 * 1 days) {
213               require((balance - amount) >= SECtotalAmount/10 * 3/4);
214
215           } else if (now < start + (2*365+1) * 1 days){
216               require((balance - amount) >= SECtotalAmount/10 * 2/4);
217
218           }else if (now < start + (3*365+1) * 1 days){
219               require((balance - amount) >= SECtotalAmount/10 * 1/4);
220
221           }
222       }
223
224       balanceOf[_from] -= _value;                         // Subtract from the sender
225       balanceOf[_to] += _value;                           // Add the same to the recipient
226       Transfer(_from, _to, _value);
227   }
```

图 3.12 缺少运算校验

安全建议：在运算前后添加 previousBalances 的计算校验，在交易前后，判断买、卖双方的数据是否和转换前一致。

4. 附录 A：合约代码

本次测试代码来源：

https://etherscan.io/address/0xc6689eb9a6d724b8d7b1d923ffd65b7005da1b62#code

```
pragma solidity ^0.4.16;
contract owned {
    address public owner;
    function owned() public {
        owner = msg.sender;
    }
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
    function transferOwnership(address newOwner) onlyOwner public {
        owner = newOwner;
    }
}
interface tokenRecipient { function receiveApproval(address _from, uint256 _value,
address _token, bytes _extraData) public; }

contract TokenERC20 {
    // Public variables of the token
    string public name;
    string public symbol;
    uint8 public decimals = 18;
    // 18 decimals is the strongly suggested default, avoid changing it
    uint256 public totalSupply;

    // This creates an array with all balances
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;
```

```
    // This generates a public event on the blockchain that will notify clients
    event Transfer(address indexed from, address indexed to, uint256 value);

    // This notifies clients about the amount burnt
    event Burn(address indexed from, uint256 value);

    /**
     * Constrctor function
     *
     * Initializes contract with initial supply tokens to the creator of the contract
     */
    function TokenERC20(
        uint256 initialSupply,
        string tokenName,
        string tokenSymbol
) public {
        totalSupply = initialSupply * 10 ** uint256(decimals);  // Update total supply
with the decimal amount
        balanceOf[msg.sender] = totalSupply;                // Give the creator all
initial tokens
        name = tokenName;                                   // Set the name for display
purposes
        symbol = tokenSymbol;                               // Set the symbol for display
purposes
    }

    /**
     * Internal transfer, only can be called by this contract
     */
    function _transfer(address _from, address _to, uint _value) internal {
        // Prevent transfer to 0x0 address. Use burn() instead
        require(_to != 0x0);
        // Check if the sender has enough
        require(balanceOf[_from] >= _value);
        // Check for overflows
        require(balanceOf[_to] + _value > balanceOf[_to]);
        // Save this for an assertion in the future
        uint previousBalances = balanceOf[_from] + balanceOf[_to];
        // Subtract from the sender
        balanceOf[_from] -= _value;
        // Add the same to the recipient
        balanceOf[_to] += _value;
        Transfer(_from, _to, _value);
        // Asserts are used to use static analysis to find bugs in your code. They
should never fail
        assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
    }

    /**
     * Transfer tokens
     *
     * Send `_value` tokens to `_to` from your account
     *
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transfer(address _to, uint256 _value) public {
        _transfer(msg.sender, _to, _value);
    }

    /**
     * Transfer tokens from other address
     *
     * Send `_value` tokens to `_to` in behalf of `_from`
     *
     * @param _from The address of the sender
     * @param _to The address of the recipient
     * @param _value the amount to send
     */
    function transferFrom(address _from, address _to, uint256 _value) public returns
(bool success) {
        require(_value <= allowance[_from][msg.sender]);     // Check allowance
        allowance[_from][msg.sender] -= _value;
        _transfer(_from, _to, _value);
        return true;
    }
```

```
    /**
     * Set allowance for other address
     *
     * Allows `_spender` to spend no more than `_value` tokens in your behalf
     *
     * @param _spender The address authorized to spend
     * @param _value the max amount they can spend
     */
    function approve(address _spender, uint256 _value) public
    returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        return true;
    }

    /**
     * Set allowance for other address and notify
     *
     * Allows `_spender` to spend no more than `_value` tokens in your behalf, and then
ping the contract about it
     *
     * @param _spender The address authorized to spend
     * @param _value the max amount they can spend
     * @param _extraData some extra information to send to the approved contract
     */
    function approveAndCall(address _spender, uint256 _value, bytes _extraData)
    public
    returns (bool success) {
        tokenRecipient spender = tokenRecipient(_spender);
        if (approve(_spender, _value)) {
            spender.receiveApproval(msg.sender, _value, this, _extraData);
            return true;
        }
    }
    /**
     * Destroy tokens
     *
     * Remove `_value` tokens from the system irreversibly
     *
     * @param _value the amount of money to burn
     */
    function burn(uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);   // Check if the sender has enough
        balanceOf[msg.sender] -= _value;            // Subtract from the sender
        totalSupply -= _value;                      // Updates totalSupply
        Burn(msg.sender, _value);
        return true;
    }

    /**
     * Destroy tokens from other account
     *
     * Remove `_value` tokens from the system irreversibly on behalf of `_from`.
     *
     * @param _from the address of the sender
     * @param _value the amount of money to burn
     */
    function burnFrom(address _from, uint256 _value) public returns (bool success) {
        require(balanceOf[_from] >= _value);                // Check if the targeted
balance is enough
        require(_value <= allowance[_from][msg.sender]);    // Check allowance
        balanceOf[_from] -= _value;                         // Subtract from the targeted
balance
        allowance[_from][msg.sender] -= _value;             // Subtract from the sender's
allowance
        totalSupply -= _value;                              // Update totalSupply
        Burn(_from, _value);
        return true;
    }
}
/******************************************/
/*       ADVANCED TOKEN STARTS HERE       */
/******************************************/
contract SECToken is owned, TokenERC20 {
    uint256 public sellPrice;
    uint256 public buyPrice;
```

```
    mapping (address => bool) public frozenAccount;
    /* This generates a public event on the blockchain that will notify clients */
    event FrozenFunds(address target, bool frozen);
    /* Initializes contract with initial supply tokens to the creator of the contract
*/
    function SECToken(
        uint256 initialSupply,
        string tokenName,
        string tokenSymbol
) TokenERC20(initialSupply, tokenName, tokenSymbol) public {}
    /* Internal transfer, only can be called by this contract */
    function _transfer(address _from, address _to, uint _value) internal {
        require (_to != 0x0);                               // Prevent transfer to 0x0
address. Use burn() instead
        require (balanceOf[_from] >= _value);               // Check if the sender has
enough
        require (balanceOf[_to] + _value > balanceOf[_to]); // Check for overflows
        require(!frozenAccount[_from]);                     // Check if sender is frozen
        require(!frozenAccount[_to]);                       // Check if recipient is frozen
        //about lock coins
        uint start = 1520956799;
        uint256 SECtotalAmount = 1500000000 * 10 ** 18;
        address teamaccount = 0xC32b1519A0d4E883FE136AbB3198cbC402b5503F;
        uint256 amount = _value;
        address sender = _from;
        uint256 balance = balanceOf[_from];
        if(teamaccount == sender){
            if (now < start + 365 * 1 days) {
                require((balance - amount) >= SECtotalAmount/10 * 3/4);
            } else if (now < start + (2*365+1) * 1 days){
                require((balance - amount) >= SECtotalAmount/10 * 2/4);
            }else if (now < start + (3*365+1) * 1 days){
                require((balance - amount) >= SECtotalAmount/10 * 1/4);
            }
        }
        balanceOf[_from] -= _value;                         // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
        Transfer(_from, _to, _value);
    }

/// @notice Create `mintedAmount` tokens and send it to `target`
    /// @param target Address to receive the tokens
    /// @param mintedAmount the amount of tokens it will receive
    function mintToken(address target, uint256 mintedAmount) onlyOwner public {
        balanceOf[target] += mintedAmount;
        totalSupply += mintedAmount;
        Transfer(0, this, mintedAmount);
        Transfer(this, target, mintedAmount);
    }

    /// @notice `freeze? Prevent | Allow` `target` from sending & receiving tokens
    /// @param target Address to be frozen
    /// @param freeze either to freeze it or not
    function freezeAccount(address target, bool freeze) onlyOwner public {
        frozenAccount[target] = freeze;
        FrozenFunds(target, freeze);
    }

    /// @notice Allow users to buy tokens for `newBuyPrice` eth and sell tokens for
`newSellPrice` eth
    /// @param newSellPrice Price the users can sell to the contract
    /// @param newBuyPrice Price users can buy from the contract
    function setPrices(uint256 newSellPrice, uint256 newBuyPrice) onlyOwner public {
        sellPrice = newSellPrice;
        buyPrice = newBuyPrice;
    }

    /// @notice Buy tokens from contract by sending ether
    function buy() payable public {
        uint amount = msg.value / buyPrice;                 // calculates the amount
        _transfer(this, msg.sender, amount);                // makes the transfers
    }

/// @notice Sell `amount` tokens to contract
   /// @param amount amount of tokens to be sold
    function sell(uint256 amount) public {
```

```
        require(this.balance >= amount * sellPrice);      // checks if the contract has
enough ether to buy
        _transfer(msg.sender, this, amount);              // makes the transfers
        msg.sender.transfer(amount * sellPrice);          // sends ether to the seller.
It's important to do this last to avoid recursion attacks
    }
}
contract SEC is SECToken(1500000000, "SEC", "SEC") {}
```

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