知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇

版本说明

修订内容	修订时间	文档编码	版本号
编写文档	2020/9/21	U1NPLVpOSFktMjAyMDA5MjEK	V1.0

文档信息

文档名称	文档版本号	文档编号	保密级别
SSO 智能合约审计报告	V1.0	SSO-ZNHY-20200921	项目组公开

声明

创宇仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇提供的文件和资料。创宇假设：已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇对由此而导致的损失和不利影响不承担任何责任。

Knownsec

目录

1. 综述 - 5 -

2. 代码漏洞分析 - 6 -

2.1. 漏洞等级分布 - 6 -

2.2. 审计结果汇总说明 - 7 -

3. 代码审计结果分析 - 8 -

3.1. 重入攻击检测【通过】 - 8 -

3.2. 数值溢出检测【通过】 - 8 -

3.3. 访问控制检测【通过】 - 9 -

3.4. 返回值调用验证【通过】 - 9 -

3.5. 错误使用随机数【通过】 - 10 -

3.6. 事务顺序依赖【低危】 - 10 -

3.7. 拒绝服务攻击【通过】 - 11 -

3.8. 逻辑设计缺陷【通过】 - 12 -

3.9. 假充值漏洞【通过】 - 12 -

3.10. 增发代币漏洞【通过】 - 12 -

3.11. 冻结账户绕过【通过】 - 13 -

4. 附录 A：合约代码 - 14 -

5. 附录 B：漏洞风险评级标准 - 15 -

6. 附录 C：漏洞测试工具简介 - 16 -

6.1. Manticore - 16 -

Knownsec

6.2. Oyente ..... - 16 -

6.3. securify.sh ..... - 16 -

6.4. Echidna ..... - 16 -

6.5. MAIAN ..... - 16 -

6.6. ethersplay ..... - 17 -

6.7. ida-evm ..... - 17 -

6.8. Remix-ide ..... - 17 -

6.9. 知道创宇渗透测试人员专用工具包 ..... - 17 -

Knownsec

1. 综述

本次报告有效测试时间是从 2020 年 9 月 19 日开始到 2020 年 9 月 21 日结束，在此期间针对 SSO **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，发现存在事物顺序依赖，但由于该漏洞利用难度大，故综合评定为通过。

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	SSO
代码类型	代币代码，波场智能合约
合约地址	TWh5Tyx68o4zKqAqyqr6mY1vXcZmL8c51m
合约链接	https://tronscan.io/#/contract/TWh5Tyx68o4zKqAqyqr6mY1vXcZmL8c51m/code
代码语言	solidity

Knownsec

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	通过
0	0	1	10

风险等级分布图

■ 高危[0个] ■ 中危[0个] ■ 低危[1个] ■ 通过[10个]

2.2. 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
智能合约	重入攻击检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	未验证返回值的调用	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	事务顺序依赖检测	低危（通过）	经检测，代码中存在事务顺序依赖风险，但由于利用难度过大，故综合评定为通过。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	逻辑设计缺陷检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	增发代币漏洞检测	通过	经检测，不存在该安全问题。
	冻结账户绕过检测	通过	经检测，不存在该安全问题。

Kn

3. 代码审计结果分析

3.1. 重入攻击检测【通过】

重入漏洞是最著名的智能合约漏洞。

在 Solidity 中，调用其他地址的函数或者给合约地址转账，都会把自己的地址作为被调合约的 msg.sender 传递过去。此时，如果传递的 energy 足够的话，就可能会被对方进行重入攻击，在波场中对合约地址调用 transfer/sender，只会传递 2300 的 Energy，这不足以发起一次重入攻击。所以要一定避免通过 call.value 的方式，调用未知的合约地址。因为 call.value 可以传入远大于 2300 的 energy。

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。

3.2. 数值溢出检测【通过】

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 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。

```
function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");
```

```
        // knownsec// 先减后加
        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
```

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【通过】

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该安全问题。

安全建议：无。

3.4. 返回值调用验证【通过】

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 trx，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 2300energy 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300energy 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 energy 进行调用（可通过传入 feeLimit 参数进行限制），不能有效防止重入攻击。

Knownsec

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 trx 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该安全问题。

安全建议：无。

3.5. 错误使用随机数【通过】

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，即这些随机数在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安全问题。

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 energy 费用，因此用户可以指定更高的费用以便更快地开展交易。由于波场区块链是公开的，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，SSO 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：

```
function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");
        //knownsec// 事务依赖顺序
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

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));
```

3.7. 拒绝服务攻击【通过】

在波场的世界中，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 energy 导致 energy 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该安全问题。

安全建议：无。

3.8. 逻辑设计缺陷【通过】

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该安全问题。

安全建议：无。

3.9. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

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。

3.10. 增发代币漏洞【通过】

检测在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中存在该问题，增发代币与交易所政策和业务逻辑相关，故给予通过。

Knownsec

```
function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");
        //knownsec// 增发代币
        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

安全建议：无。

3.11. 冻结账户绕过【通过】

检测代币合约中在转移代币时，是否存在未校验代币来源账户、发起账户、目标账户是否被冻结的操作。

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。

Knownsec

4. 附录 A：合约代码

本次测试代码来源：

```
MineableToken.sol

pragma solidity ^0.5.9;

import "./ERC20.sol";
import "./ERC20Detailed.sol";
import "./IMineableToken.sol";

contract MineableToken is IMineableToken, ERC20, ERC20Detailed, Ownable {
    using SafeMath for uint256;

    uint256 public totalSupplyCap;

    constructor(
        address _owner,
        address _minter,
        string memory _name,
        string memory _symbol,
        uint256 _supplyCap,
        uint8 _decimals)

    public ERC20Detailed(_name, _symbol, _decimals) Ownable(_owner){
        minters[_minter] = _minter;
        if(_supplyCap == 0) {
            totalSupplyCap = ~uint256(0);
        } else{
            totalSupplyCap = _supplyCap;
        }
    }

    mapping(address=>address) public minters;

    function setMinter(address _minter) public {
        require(isMinter(msg.sender) || isOwner());
        minters[_minter] = _minter;
    }

    function removeMinter(address _minter) public {
        require(isMinter(msg.sender) || isOwner());
        delete minters[_minter];
    }

    function isMinter(address _minter) public view returns(bool) {
        return (address(0) != minters[_minter]);
    }

    function setTotalSupplyCap(uint256 newCap) public onlyOwner {
        uint256 currentSupply = totalSupply();
        require(newCap > currentSupply);

        totalSupplyCap = newCap;
    }

    function mint(address to, uint256 amount) public {
        require(minters[msg.sender] != address(0), "not minter");

        uint256 currentSupply = totalSupply();
        require(currentSupply < totalSupplyCap, "total supply cap touched");

        uint256 afterMint = currentSupply.add(amount);

        if(afterMint > totalSupplyCap) {
            amount = totalSupplyCap.sub(currentSupply);
        }

        _mint(to, amount);
    }

    function signature() external pure returns (string memory) {
        return "provided by SSO-SC / www.SSOsc.com";
    }
}
```

Knownsec

5. 附录 B：漏洞风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 TRX 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 TRX 导致的拒绝服务漏洞、因 energy 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 TRX 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 energy 触发的事务顺序依赖风险等。

Knownsec

6. 附录 C：漏洞测试工具简介

6.1. 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。

6.2. 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。

6.3. securify.sh

Securify 可以验证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。

6.4. Echidna

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

6.5. MAIAN

MAIAN 是一个用于查找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。

Knownsec

6.6. ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7. ida-evm

ida-evm 是一个针对虚拟机（EVM）的 IDA 处理器模块。

6.8. Remix-ide

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

6.9. 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。

Knownsec

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509