

智能合约审计报告

YT



主测人: 何平生

版本说明

修订人	修订内容	修订时间	版本号	审阅人
何平生	编写文档	2019/01/29	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
YT 智能合约审计报告	V1.0	【YT-DMSJ-20190129】	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。

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【安全】	8 -
3.2. 数值溢出检测【安全】	8 -
3.3. 访问控制检测【安全】	9 -
3.4. 返回值调用验证【安全】	9 -
3.5. 错误使用随机数【安全】	9 -
3.6. 事务顺序依赖【安全】	10 -
3.7. 拒绝服务攻击【安全】	10 -
3.8. 逻辑设计缺陷【安全】	11 -
3.9. 假充值漏洞【安全】	11 -
3.10. 增发代币漏洞【安全】	11 -
3.11. 冻结账户绕过【安全】	11 -
4. 附录 A：合约代码.....	13 -
5. 附录 B：漏洞风险评级标准	18 -
6. 附录 C：漏洞测试工具简介	19 -
6.1. Manticore.....	19 -

6.2.	Oyente.....	- 19 -
6.3.	securify.sh.....	- 19 -
6.4.	Echidna.....	- 19 -
6.5.	MAIAN.....	- 19 -
6.6.	ethersplay.....	- 20 -
6.7.	ida-evm.....	- 20 -
6.8.	Remix-ide.....	- 20 -
6.9.	知道创宇渗透测试人员专用工具包.....	- 20 -

KNOWNSEC

1. 综述

本次报告有效测试时间是从 2019 年 01 月 29 日开始到 2019 年 01 月 29 日结束，在此期间针对 YT 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。

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	CherryToken
代码类型	代币代码
合约地址	0x5c89736e9454200141b80c37eb28eaceca2ce8cb
链接地址	https://etherscan.io/address/0x5c89736e9454200141b80c37eb28eaceca2ce8cb#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
何平生	heps@knownsec.com	高级工程师

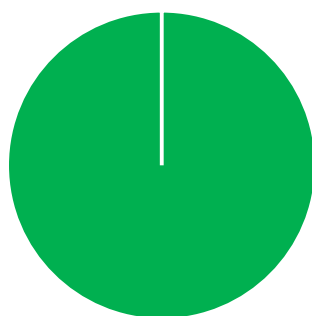
2. 代码漏洞分析

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	通过
0	0	0	11

风险等级分布图



■ 高危[0个] ■ 中危[0个] ■ 低危[0个] ■ 通过[11个]

2.2. 审计结果汇总说明

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	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	通过	经检测，不存在该安全问题。
	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。

综合评定结果：通过

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 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。

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该安全问题。

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问题。

安全建议：无。

3.6. 事务顺序依赖【安全】

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，智能合约代码中不存在该问题。

安全建议：无。

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该问题。

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。

3.9. 假充值漏洞【安全】

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问题。

安全建议：无。

3.10. 增发代币漏洞【安全】

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，智能合约代码中不存在该问题。

安全建议：无。

3.11. 冻结账户绕过【安全】

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问题。

安全建议：无。

Knownsec

4. 附录 A：合约代码

本次测试代码来源：

```

CherryToken.sol
pragma solidity ^0.4.24; //knownsec//指定编译器版本, 符合推荐做法
//knownsec//使用 SafeMath 函数进行数值运算操作, 符合推荐做法

library SafeMath {
    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;
    }

    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;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        assert(b <= a);
        return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }
}

//knownsec//ERC20 接口
contract ERC20 {

    function name() public view returns (string);
    function symbol() public view returns (string);
    function decimals() public view returns (uint8);

    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}

//knownsec//ERC233 接口
contract ERC223 {
    function transferdata(address to, uint value, bytes data) payable public;
    event Transferdata(address indexed from, address indexed to, uint value, bytes indexed data);
}

//knownsec//ERC233 自定义回调函数
contract ERC223ReceivingContract {
    function tokenFallback(address _from, uint _value, bytes _data) public;
}

contract ERCAddressFrozenFund is ERC20{

    using SafeMath for uint; //knownsec//通过这种方式引用 SafeMath 更方便

    struct LockedWallet { //knownsec//定义账户_资金结构体数据类型
        address owner; // the owner of the locked wallet, he/she must secure the private key
        uint256 amount;
        uint256 start; // timestamp when "lock" function is executed
        uint256 duration; // duration period in seconds. if we want to lock an amount for
        uint256 release; // release = start+duration
        // "start" and "duration" is for bookkeeping purpose only. Only "release" will be actually checked once
        unlock function is called
    }
}

```

```

address public owner;
uint256 _lockedSupply;

mapping (address => LockedWallet) addressFrozenFund; //address -> (deadline, amount), freeze fund of an
address its so that no token can be transferred out until deadline

function mintToken(address _owner, uint256 amount) internal;
function burnToken(address _owner, uint256 amount) internal;

event LockBalance(address indexed addressOwner, uint256 releasetime, uint256 amount);
event LockSubBalance(address indexed addressOwner, uint256 index, uint256 releasetime, uint256 amount);
event UnlockBalance(address indexed addressOwner, uint256 releasetime, uint256 amount);
event UnlockSubBalance(address indexed addressOwner, uint256 index, uint256 releasetime, uint256 amount);

function lockedSupply() public view returns (uint256) {
    return _lockedSupply;
}

function releaseTimeOf(address _owner) public view returns (uint256 releaseTime) {
    return addressFrozenFund[_owner].release;
}

function lockedBalanceOf(address _owner) public view returns (uint256 lockedBalance) {
    return addressFrozenFund[_owner].amount;
}

function lockBalance(uint256 duration, uint256 amount) public{//knownsec//锁定（冻结）资金
    address _owner = msg.sender;//knownsec//将msg.sender 传给_owner，之后对_owner 进行操作，故调用者只可以冻结自己的资金
    require(address(0) != _owner && amount > 0 && duration > 0 && balanceOf(_owner) >= amount);//knownsec//检查地址是否为空，以及amount 的有效性和duration 的有效性、balance 是否足够等
    require(addressFrozenFund[_owner].release <= now && addressFrozenFund[_owner].amount == 0);
    //knownsec//更新_owner 的资产结构体数据类型内的数据
    addressFrozenFund[_owner].start = now;
    addressFrozenFund[_owner].duration = duration;
    addressFrozenFund[_owner].release = SafeMath.add(addressFrozenFund[_owner].start, duration);
    addressFrozenFund[_owner].amount = amount;
    burnToken(_owner, amount);//knownsec//调用函数销毁token，调用者只可以销毁自己的token
    lockedSupply = SafeMath.add(_lockedSupply, lockedBalanceOf(_owner));//knownsec//更新冻结的token 的数量
    emit LockBalance(_owner, addressFrozenFund[_owner].release, amount);
}

//_owner must call this function explicitly to release locked balance in a locked wallet
function releaseLockedBalance() public{//knownsec//解锁之前冻结的token
    address _owner = msg.sender;//knownsec//将msg.sender 传递给_owner，之后对_owner 进行操作，故只可以解锁自己之前冻结的token
    require(address(0) != _owner && lockedBalanceOf(_owner) > 0 && releaseTimeOf(_owner) <= now);//knownsec//检查地址是否为空、冻结的balance 是否满足条件等
    mintToken(_owner, lockedBalanceOf(_owner));//knownsec//解锁之前冻结的token
    lockedSupply = SafeMath.sub(_lockedSupply, lockedBalanceOf(_owner));//knownsec//更新冻结的token 的数量
    emit UnlockBalance(_owner, addressFrozenFund[_owner].release, lockedBalanceOf(_owner));
    delete addressFrozenFund[_owner];//knownsec//从冻结账户结构体表中删除当前账户
}
}

contract CherryToken is ERC223, ERCAddressFrozenFund {
    using SafeMath for uint;//knownsec//通过这种方式引用SafeMath 更方便

    string internal _name;
    string internal _symbol;
    uint8 internal _decimals;
    uint256 internal _totalSupply;
    address public fundsWallet;
    uint256 internal fundsWalletChanged;

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

    //knownsec//构造函数，初始化
    constructor() public {
        _symbol = "YT";
        _name = "Cherry Token";
        _decimals = 8;
        _totalSupply = 1000000000000000;
    }
}

```

```

balances[msg.sender] = _totalSupply;
fundsWallet = msg.sender;

owner = msg.sender;

fundsWalletChanged = 0;
}

function changeFundsWallet(address newOwner) public{//knownsec//钱包变动
require(msg.sender == fundsWallet && fundsWalletChanged == 0);//knownsec//判断当前处理者是否有当前的操作权限(只有用户自己才可以对自己的钱包进行操作)及是否已经变动过钱包
//knownsec//此处建议增加对 newOwner 地址是否为空的判断: require(newOwner!=address(0));
balances[newOwner] = balances[fundsWallet];//knownsec//将原 owner 的 balance 赋值给新 owner
balances[fundsWallet] = 0;//knownsec//将原 owner 的 balance 置为空
fundsWallet = newOwner;//knownsec//变更钱包的账户
fundsWalletChanged = 1;//knownsec//设置钱包变更状态
}

function name() public view returns (string) {
return _name;
}

function symbol() public view returns (string) {
return _symbol;
}

function decimals() public view returns (uint8) {
return _decimals;
}

function totalSupply() public view returns (uint256) {
return _totalSupply;
}

function mintToken(address owner, uint256 amount) internal{//knownsec//该函数只能内部调用, 同时对于调用的_owner 都有检查, 只能操作者自己处理自己的资产
balances[_owner] = SafeMath.add(balances[_owner], amount);
}

function burnToken(address owner, uint256 amount) internal{//knownsec//该函数只能内部调用, 同时对于调用的_owner 都有检查, 只能操作者自己处理自己的资产
balances[_owner] = SafeMath.sub(balances[_owner], amount);
}

function() payable public {
require(msg.sender == address(0));//disable ICO crowd sale 禁止 ICO 资金募集, 因为本合约已经过了募集阶段
}

function transfer(address to, uint256 _value) public returns (bool) {
require(to != address(0));//knownsec//检查地址是否为空
require(_value <= balances[msg.sender]);//knownsec//检查用户是否有足够的 balance 用于转账

if(isContract(to))){//knownsec//检查是否为一个合约地址
ERC223ReceivingContract receiver = ERC223ReceivingContract(to);
bytes memory data = new bytes(1);
receiver.tokenFallback(msg.sender, _value, data);
}

balances[msg.sender] = SafeMath.sub(balances[msg.sender], _value);//knownsec//使用 SafeMath 函数先减后加符合推荐操作
balances[to] = SafeMath.add(balances[to], _value);
emit Transfer(msg.sender, to, _value);

return true;
}

function balanceOf(address owner) public view returns (uint256 balance) {
return balances[_owner];
}

function transferFrom(address from, address to, uint256 _value) public returns (bool) {
require(to != address(0));//knownsec//检查地址是否为空
require(_value <= balances[from]);//knownsec//检查是否有足够的 balance 用于转账
require(_value <= allowed[from][msg.sender]);//knownsec//检查授权的代为转账的额度是否大于当前要转账的额度

if(from == fundsWallet){//knownsec//判断from 账户是否为 fundsWallet
require(_value <= balances[from]);
}

if(isContract(to))){//knownsec//检查地址是否为一个合约地址
ERC223ReceivingContract receiver = ERC223ReceivingContract(to);
bytes memory data = new bytes(1);
receiver.tokenFallback(msg.sender, _value, data);
}
}

```

```

balances[_from] = SafeMath.sub(balances[_from], _value); //knownsec//先减后加符合推荐操作
balances[_to] = SafeMath.add(balances[_to], _value);
allowed[_from][msg.sender] = SafeMath.sub(allowed[_from][msg.sender], _value); //knownsec//更新授
权代为转账的额度

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

function approve(address _spender, uint256 _value) public returns (bool) {
    require((_value == 0) || (allowed[msg.sender][_spender] == 0)); //knownsec//防止“事务顺序依赖问
题”
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}

function allowance(address _owner, address _spender) public view returns (uint256) {
    return allowed[_owner][_spender];
}

function increaseApproval(address _spender, uint _addedValue) public returns (bool) {
    allowed[msg.sender][_spender] = SafeMath.add(allowed[msg.sender][_spender], _addedValue);
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
}

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] = SafeMath.sub(oldValue, _subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
}
//knownsec//处理单笔交易
function transferdata(address _to, uint _value, bytes _data) public payable {
    require(_value > 0);
    if(isContract(_to)) //knownsec//检查是否为一个合约地址
        ERC223ReceivingContract receiver = ERC223ReceivingContract(_to);
        receiver.tokenFallback(msg.sender, _value, _data);
}

balances[msg.sender] = balances[msg.sender].sub(_value); //knownsec//先减后加(使用SafeMath函数
操作)
balances[_to] = balances[_to].add(_value);

    emit Transferdata(msg.sender, _to, _value, _data);
}

function isContract(address _addr) private view returns (bool is_contract) { //knownsec//判断一个地址是否
为合约地址
    uint length;
    assembly {
        //retrieve the size of the code on target address, this needs assembly
        length := extcodesize(_addr)
    }
    return (length>0);
}

function transferMultiple(address[] _tos, uint256[] _values, uint count) payable public returns (bool)
{//knownsec//处理多笔交易
    uint256 total = 0;
    uint256 total_prev = 0;
    uint i = 0;

    for(i=0;i<count;i++){ //knownsec//uint 默认为uint256,数值较大,而且这里循环的依据应该是地址的
个数,而不应该是一个可控的变量,根据变量去循环,建议使用地址的个数来循环(_tos.length),同时对地
址的个数做一个限制
        require(_tos[i] != address(0) && !isContract(_tos[i])); // _tos must no contain any contract address

        if(isContract(_tos[i])) {
            ERC223ReceivingContract receiver = ERC223ReceivingContract(_tos[i]);
            bytes memory _data = new bytes(1);
            receiver.tokenFallback(msg.sender, _values[i], _data);
        }
        //knownsec//计算要转账的总额度
        total_prev = total;
        total = SafeMath.add(total, _values[i]);
        require(total >= total_prev);
    }

    require(total <= balances[msg.sender]); //knownsec//检查是否有足够的转账额度
    //knownsec//进行转账操作
}

```



```
for(i=0;i<count;i++){//knownsec//uint 默认为uint256,数值较大,而且这里循环的依据应该是地址的  
个数,而不应该是一个可控的变量,根据变量去循环,建议使用地址的个数来循环(_tos.length),同时对地址  
的个数做一个限制  
    balances[msg.sender] = SafeMath.sub(balances[msg.sender], values[i]);//knownsec//先减后加  
    balances[_tos[i]] = SafeMath.add(balances[_tos[i]], _values[i]);  
    emit Transfer(msg.sender, _tos[i], _values[i]);  
}  
return true;  
}
```

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	<p>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 ETH 或代币的重入漏洞等；</p> <p>能造成代币合约归属感丢失的漏洞，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；</p> <p>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 ETH 导致的拒绝服务漏洞、因 gas 耗尽导致的拒绝服务漏洞。</p>
中危漏洞	<p>需要特定地址才能触发的高风险漏洞，如代币合约所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。</p>
低危漏洞	<p>难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 ETH 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 gas 触发的事务顺序依赖风险等。</p>

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。

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。

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。

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 是一个针对以太坊虚拟机（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 语言构建以太坊合约并调试交易。

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。



【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