知道创宇

密级 ★ | 项目组公开

智能合约安全审计报告

审计结果

通过

★ ★ ★ ★ ★

版本说明

修订人	修订内容	修订时间	版本号	审阅人
罗逸锋	编写文档	2020/10/15	V1.0	徐昊杰

文档信息

文档名称	审计日期	审计结果	保密级别	审计查询电话
Stake 智能合约安全审计报告	2020/10/15	通过	项目组公开	400-060-9587

版权声明

公司声明

北京知道创宇信息技术股份有限公司基于项目方截至本报告出具时向我方提供的文件和资料，仅对该项目的安全情况进行约定内的安全审计并出具了本报告，我方无法对该项目的背景、项目的实用性、商业模式的合规性、以及项目的合法性等其他情况进行风险判断，亦不对此承担责任。该报告仅供项目方内部决策参考使用，未经我方书面同意，不得擅自将报告予以公开或者提供给其他人或者用于其他目的，我方出具的报告不得作为第三方的任何行为和决策的依据，我方不对用户及第三方因报告采取的相关决策产生的后果承担任何责任。我方假设：截至本报告出具时项目方向我方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，我方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

本次报告有效测试时间是从 2020 年 10 月 13 日开始到 2020 年 10 月 15 日结束，在此期间针对 Xfinance 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，未发现中、高危安全风险，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：

项目名称	项目内容
Token 名称	Stake
代码类型	代币代码
代码语言	Solidity
代码地址	https://etherscan.io/address/0x5BEfBB272290dD5b8521D4a938f6c4757742c430#code

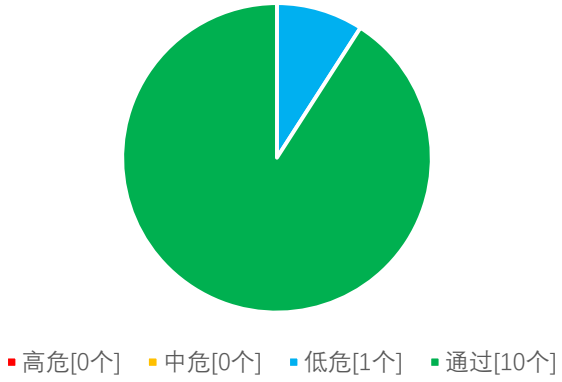
2. 代码漏洞分析

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



2.2. 审计结果汇总

（其他未知安全漏洞不包含在本次审计责任范围）

审计结果			
测试项目	测试内容	状态	描述
智能合约安全审计	**重入攻击检测**	通过	检查 call.value()函数使用安全
	数值溢出检测	通过	检查 add 和 sub 函数使用安全
	访问控制缺陷检测	通过	检查各操作访问权限控制
	未验证返回值的调用	通过	检查转币方法看是否验证返回值
	错误使用随机数检测	通过	检查是否具备统一的内容过滤器
	事务顺序依赖检测	通过	检查是否存在事务顺序依赖风险
	拒绝服务攻击检测	低危	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	通过	检查智能合约代码中与业务设计相关的安全问题
	假充值漏洞检测	通过	检查智能合约代码中是否存在假充值漏洞
	增发代币漏洞检测	通过	检查智能合约中是否存在增发代币的功能
	冻结账户绕过检测	通过	检查转移代币中是否存在未校验冻结账户的问题

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区块链智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送代币的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送代币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。

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代币

，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，可能由于代币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【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【通过】

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相关漏洞。

安全建议：无。

3.7. 拒绝服务攻击【低危】

在区块链的世界中，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，智能合约代码中存在该问题。

```
197     function transferOwnership(address payable _newOwner) public onlyOwner {
198         owner = _newOwner;
199         emit OwnershipTransferred(msg.sender, _newOwner);
200     }
201 }
```

安全建议：对于控制权限的转换需要注意对于用户所有权的确定，避免造成控制权的永久丢失。

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相关漏洞。

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(ABBT.sender)的余额检查用的是 if 判断方式，当 balances[ABBT.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相关漏洞。

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，智能合约代码中不存在相关漏洞。

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问题。

安全建议：无。

4. 附录 A：合约代码

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-12
*/

/**
 *Submitted for verification at Etherscan.io on 2020-08-29
*/

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.6.0;

// ----------------------------------------------------------------------------
// 'XFI' Staking smart contract
// ----------------------------------------------------------------------------

// ----------------------------------------------------------------------------
// SafeMath library
// ----------------------------------------------------------------------------

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom
message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
```

```
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom
message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
modulo),
```

```
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }

    function ceil(uint a, uint m) internal pure returns (uint r) {
        return (a + m - 1) / m * m;
    }
}

// ----------------------------------------------------------------------------
// Owned contract
// ----------------------------------------------------------------------------
contract Owned {
    address payable public owner;

    event OwnershipTransferred(address indexed _from, address indexed _to);

    constructor() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function transferOwnership(address payable _newOwner) public onlyOwner {
        owner = _newOwner;
        emit OwnershipTransferred(msg.sender, _newOwner);
    }
}

// ----------------------------------------------------------------------------
// ERC Token Standard #20 Interface
// ----------------------------------------------------------------------------
interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address tokenOwner) external view returns (uint256 balance);
    function allowance(address tokenOwner, address spender) external view returns
(uint256 remaining);
    function transfer(address to, uint256 tokens) external returns (bool success);
    function approve(address spender, uint256 tokens) external returns (bool success);
    function transferFrom(address from, address to, uint256 tokens) external returns
(bool success);
    function burnTokens(uint256 _amount) external;
    event Transfer(address indexed from, address indexed to, uint256 tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint256
tokens);
```

```
}

// ----------------------------------------------------------------------------
// ERC20 Token, with the addition of symbol, name and decimals and assisted
// token transfers
// ----------------------------------------------------------------------------
contract Stake is Owned {
    using SafeMath for uint256;

    address public XFI = 0x5BEfBB272290dD5b8521D4a938f6c4757742c430;

    uint256 public totalStakes = 0;
    uint256 stakingFee = 25; // 2.5%
    uint256 unstakingFee = 25; // 2.5%
    uint256 public totalDividends = 0;
    uint256 private scaledRemainder = 0;
    uint256 private scaling = uint256(10) ** 12;
    uint public round = 1;

    struct USER{
        uint256 stakedTokens;
        uint256 lastDividends;
        uint256 fromTotalDividend;
        uint round;
        uint256 remainder;
    }

    mapping(address => USER) stakers;
    mapping (uint => uint256) public payouts;              // keeps record of each
payout

    event STAKED(address staker, uint256 tokens, uint256 stakingFee);
    event UNSTAKED(address staker, uint256 tokens, uint256 unstakingFee);
    event PAYOUT(uint256 round, uint256 tokens, address sender);
    event CLAIMEDREWARD(address staker, uint256 reward);

    // ------------------------------------------------------------------------
    // Token holders can stake their tokens using this function
    // @param tokens number of tokens to stake
    // ------------------------------------------------------------------------
    function STAKE(uint256 tokens) external {
        require(IERC20(XFI).transferFrom(msg.sender, address(this), tokens), "Tokens
cannot be transferred from user account");

        uint256 _stakingFee = 0;
        if(totalStakes > 0)
            _stakingFee= (onePercent(tokens).mul(stakingFee)).div(10);

        if(totalStakes > 0)
            // distribute the staking fee accumulated before updating the user's stake
            _addPayout(_stakingFee);

        // add pending rewards to remainder to be claimed by user later, if there is
any existing stake
        uint256 owing = pendingReward(msg.sender);
        stakers[msg.sender].remainder += owing;

        stakers[msg.sender].stakedTokens =
(tokens.sub(_stakingFee)).add(stakers[msg.sender].stakedTokens);
        stakers[msg.sender].lastDividends = owing;
        stakers[msg.sender].fromTotalDividend= totalDividends;
        stakers[msg.sender].round =  round;

        totalStakes = totalStakes.add(tokens.sub(_stakingFee));

        emit STAKED(msg.sender, tokens.sub(_stakingFee), _stakingFee);
    }

    // ------------------------------------------------------------------------
    // Owners can send the funds to be distributed to stakers using this function
    // @param tokens number of tokens to distribute
    // ------------------------------------------------------------------------
    function ADDFUNDS(uint256 tokens) external {
        require(IERC20(XFI).transferFrom(msg.sender, address(this), tokens), "Tokens
cannot be transferred from funder account");
        _addPayout(tokens);
    }
```

```
    // ------------------------------------------------------------------------
    // Private function to register payouts
    // ------------------------------------------------------------------------
    function _addPayout(uint256 tokens) private{
        // divide the funds among the currently staked tokens
        // scale the deposit and add the previous remainder
        uint256 available = (tokens.mul(scaling)).add(scaledRemainder);
        uint256 dividendPerToken = available.div(totalStakes);
        scaledRemainder = available.mod(totalStakes);

        totalDividends = totalDividends.add(dividendPerToken);
        payouts[round] = payouts[round-1].add(dividendPerToken);

        emit PAYOUT(round, tokens, msg.sender);
        round++;
    }

    // ------------------------------------------------------------------------
    // Stakers can claim their pending rewards using this function
    // ------------------------------------------------------------------------
    function CLAIMREWARD() public {
        if(totalDividends > stakers[msg.sender].fromTotalDividend){
            uint256 owing = pendingReward(msg.sender);

            owing = owing.add(stakers[msg.sender].remainder);
            stakers[msg.sender].remainder = 0;

            require(IERC20(XFI).transfer(msg.sender,owing), "ERROR: error in sending
reward from contract");

            emit CLAIMEDREWARD(msg.sender, owing);

            stakers[msg.sender].lastDividends = owing; // unscaled
            stakers[msg.sender].round = round; // update the round
            stakers[msg.sender].fromTotalDividend = totalDividends; // scaled
        }
    }

    // ------------------------------------------------------------------------
    // Get the pending rewards of the staker
    // @param _staker the address of the staker
    // ------------------------------------------------------------------------
    function pendingReward(address staker) private returns (uint256) {
        uint256 amount =  ((totalDividends.sub(payouts[stakers[staker].round -
1])).mul(stakers[staker].stakedTokens)).div(scaling);
        stakers[staker].remainder += ((totalDividends.sub(payouts[stakers[staker].round
- 1])).mul(stakers[staker].stakedTokens)) % scaling ;
        return amount;
    }

    function getPendingReward(address staker) public view returns(uint256
_pendingReward) {
        uint256 amount =  ((totalDividends.sub(payouts[stakers[staker].round -
1])).mul(stakers[staker].stakedTokens)).div(scaling);
        amount += ((totalDividends.sub(payouts[stakers[staker].round -
1])).mul(stakers[staker].stakedTokens)) % scaling ;
        return (amount + stakers[staker].remainder);
    }

    // ------------------------------------------------------------------------
    // Stakers can un stake the staked tokens using this function
    // @param tokens the number of tokens to withdraw
    // ------------------------------------------------------------------------
    function WITHDRAW(uint256 tokens) external {

        require(stakers[msg.sender].stakedTokens >= tokens && tokens > 0, "Invalid
token amount to withdraw");

        uint256 _unstakingFee = (onePercent(tokens).mul(unstakingFee)).div(10);

        // add pending rewards to remainder to be claimed by user later, if there is
any existing stake
        uint256 owing = pendingReward(msg.sender);
        stakers[msg.sender].remainder += owing;
```

```
        require(IERC20(XFI).transfer(msg.sender, tokens.sub(_unstakingFee)), "Error in
un-staking tokens");

        stakers[msg.sender].stakedTokens =
stakers[msg.sender].stakedTokens.sub(tokens);
        stakers[msg.sender].lastDividends = owing;
        stakers[msg.sender].fromTotalDividend= totalDividends;
        stakers[msg.sender].round =  round;

        totalStakes = totalStakes.sub(tokens);

        if(totalStakes > 0)
            // distribute the un staking fee accumulated after updating the user's stake
            _addPayout(_unstakingFee);

        emit UNSTAKED(msg.sender, tokens.sub(_unstakingFee), _unstakingFee);
    }

    // -----------------------------------------------------------------------
    // Private function to calculate 1% percentage
    // -----------------------------------------------------------------------
    function onePercent(uint256 _tokens) private pure returns (uint256){
        uint256 roundValue = _tokens.ceil(100);
        uint onePercentofTokens = roundValue.mul(100).div(100 * 10**uint(2));
        return onePercentofTokens;
    }

    // -----------------------------------------------------------------------
    // Get the number of tokens staked by a staker
    // @param _staker the address of the staker
    // -----------------------------------------------------------------------
    function yourStakedXFI(address staker) external view returns(uint256 stakedXFI){
        return stakers[staker].stakedTokens;
    }

    // -----------------------------------------------------------------------
    // Get the XFI balance of the token holder
    // @param user the address of the token holder
    // -----------------------------------------------------------------------
    function yourXFIBalance(address user) external view returns(uint256 XFIBalance){
        return IERC20(XFI).balanceOf(user);
    }
}
```

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 ETH 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 ETH 导致的拒绝服务漏洞、因 gas 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 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。

6. 附录 C：漏洞测试工具简介

6.1. MaABBTicore

MaABBTicore 是一个分析二进制文件和智能合约的符号执行工具，MaABBTicore 包含一个符号区块链虚拟机（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，MaABBTicore 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. OyeABBTe

OyeABBTe 是一个智能合约分析工具，OyeABBTe 可以用来检测智能合约中常见的 bug，比如 reeABBTrancy、事务排序依赖等等。更方便的是，OyeABBTe 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 可以验证区块链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 是一个用于查找区块链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 是一个针对区块链虚拟机（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 语言构建区块链合约并调试交易。

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。

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