猎豹区块链安全中心

智能合约审计报告

Wallet2/3 多签钱包、

Wallet m/n 多签钱包项目

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
王海龙	编写文档	2019 年 4 月 15 日	V1.0	隋欣

文档信息

文档名称	文档版本号	文档编号	保密级别
智能合约审计报告	V1.0	0x46c3f2da(23contract.sol) 0xac403885(mncontract.sol)	保密

版权声明

目录

一、 综述

1.1 审计背景

Cheetah Mobile Security 团队于 2019 年 4 月 15 日 对 Wallet 2/3 多签合约、Wallet m/n 多签合约项目的智能合约进行了安全审计。本报告详细阐述了审计细节与结果。

（**声明**：Cheetah Mobile Security 仅根据本报告出具前已经发生或存在的风险漏洞实情出具报告，并就此承担相应责任。对于报告出具以后发生或存在的事实，无法判断其对该项目造成的影响，Cheetah Mobile Security 并不对此承担责任。本报告所做的安全审计分析及其他内容，仅基于信息提供者截止本报告出具时向 Cheetah Mobile Security 提供的文件和资料出具报告，资料提供方有义务保证已提供资料不存在缺失、篡改、删减或隐瞒的情况。如果存在，Cheetah Mobile Security 对此而导致的损失和不利影响不承担任何责任。）

1.2 审计结果

审计结果	审计人	审阅人
通过	王海龙	隋欣

二、合约代码漏洞分析

2.1 合约代码漏洞等级

合约漏洞数量按等级统计如下：

高危漏洞	中危漏洞	低危漏洞
0	0	0

2.2 合约代码漏洞分布

合约漏洞按类型分布如下：（无）

2.3 合约代码审计项及结果

（其他未知安全漏洞不包含在本次审计责任范围）

审计方法	审计大类	审计子类	审计结果
攻防类审计	算数安全审计	整数上溢审计	通过
		整数下溢审计	通过
		运算精度审计	通过
	竞态审计	重入攻击审计	通过
		事物顺序依赖审计	通过
	权限控制审计	权限漏洞审计	通过
		权限过大审计	通过
	安全设计审计	安全模块使用安全	通过
		编译器版本安全	通过
		硬编码地址安全	通过
		敏感函数（fallback/call/tx.origin）使用安全	通过
		函数返回值安全	通过
	拒绝服务审计	–	通过
	Gas 优化审计	–	通过
	设计逻辑审计	–	通过

二、 代码审计结果分析

3.1 算数安全审计

智能合约中的算数安全审计分为三部分：整数上溢审计、整数下溢审计以及运算精度设计。

3.1.1 整数上溢出审计【通过】

Solidity 最多能处理 256 位的数据。当最大数字增加会上溢。整数上溢如果发生在转账逻辑中，会使得转账资金数额运算错误，导致严重的资金风险。

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

3.1.2 整数下溢出审计【通过】

Solidity 最多能处理 256 位的数据。当最小数字减小会下溢。整数下溢如果发生在转账逻辑中，会使得转账资金数额运算错误，导致严重的资金风险。

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

3.1.3 运算精度审计【通过】

solidity 在做乘法、除法的运算的过程中，会进行类型强制转换。例如整数相除，若有余数，则会丢失精度。如果资金变量运算中包含了精度风险，则会导

致用户转账逻辑错误，资金损失。

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 竞态竞争审计

智能合约中的竞态审计分为两部分：重入攻击审计以及事物顺序依赖审计。具有竞态竞争漏洞的合约中，攻击者可以在一定概率下通过调整程序或交易的执行过程，修改程序的输出结果。

3.2.1 重入攻击审计【通过】

重入攻击指当调用 call.value()函数发送 Ether 时候，攻击者使得函数发送 Ether 的操作发生在实际减少发送者账户的余额之前，从而导致异常转账。如经典的 THE DAO 攻击。

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

3.2.2 事物顺序依赖审计【通过】

事物顺序依赖指，在以太坊中，矿工在打包交易的时候，会根据交易的手续 gas 费用的高低进行排序。攻击者可以故意提高其攻击交易的 gas 费，从而使得攻击交易打包在合法交易之前，从而导致异常转账。

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 权限控制审计

智能合约中的权限控制审计分为两部分：权限漏洞审计以及权限过大审计。

3.3.1 权限漏洞审计【通过】

在有权限漏洞的智能合约中，攻击者可以对自身的账户进行提权，从而获得更高的执行权限，实施越权攻击。

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。

3.3.2 权限过大审计【通过】

权限过大审计则着眼于审计合约中是否存在特殊用户权限过大，如导致无限增发代币等情况。

检测结果：经检测，智能合约代码中不存在该安全问题。

3.4 安全设计审计

智能合约中的安全设计审计分为五部分：安全模块使用、编译器版本安全、硬编码地址安全、敏感函数使用安全以及函数返回值安全。

3.4.1 安全模块使用【通过】

安全模块使用审计会审计目标代码是否使用如 OpenZepplin 提供的 SafeMath 库函数，以避免溢出漏洞的产生；如未使用此类库，则审计代码执行过程中是否严格校验转账金额大小，以避免溢出漏洞的产生。

检测结果：经检测，智能合约代码中使用了 SafeMath 库。

安全建议：无

3.4.2 编译器版本安全审计【通过】

编译器版本安全着眼于审计代码是否明确地指出编译器版本，并审计是否使用的编译器版本过低，导致异常状况无法抛出。

检测结果：经检测，智能合约代码编译器版本> 0.4.0，当转入 Ether 无 fallback 函数时，会抛出异常。

安全建议：无

3.4.3 硬编码地址安全审计【通过】

硬编码地址安全分析代码中硬编码的地址，可能是外部合约调用地址、个人地址，审计外部合约是否存在异常从而影响本合约的执行。

检测结果：经检测，智能合约代码未采用硬编码。

安全建议：无

3.4.4 敏感函数使用安全审计【通过】

敏感函数使用主要分析合约代码中是否使用了 fallback、call、tx.origin 等不推荐的函数。

检测结果：经检测，智能合约代码未使用敏感函数。

安全建议：无

3.4.5 函数返回值安全审计【通过】

函数返回值设计主要分析函数是否正确地抛出了异常、是否正确地返回了交易的状态，避免“假充值”漏洞。

检测结果：经检测，智能合约代码能够正确抛出交易异常，返回交易状态，返回值安全。

安全建议：无

3.5 拒绝服务审计【通过】

拒绝服务攻击可能导致智能合约永远无法恢复正常工作状态。攻击手段分为：交易接收方的恶意行为、人为增加计算功能、滥用合约 private 组件等。

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

3.6 Gas 优化审计【通过】

如果智能合约中某个函数的计算量过大，如通过循环向变长数组进行批量转账，则极易造成交易的 gas 费过高，甚至达到区块的 gasLimit 导致交易执行失败。

检测结果：经检测，智能合约代码不存在该安全问题。

安全建议：无。

3.7 设计逻辑审计【通过】

除攻击漏洞外，代码在实现业务的过程中，逻辑存在问题，从而导致执行结果异常。

检测结果：经检测，智能合约代码不存在该安全问题。

安全建议：无。

附录一：合约源码审计详情

1、23contract.sol

```
pragma solidity ^0.5.4;  //Cheetah Mobile Security//指定编译器版本，符合安全编码规范
contract ERC20Interface {
    function transfer(address _to, uint256 _value) public returns (bool success);
    function balanceOf(address _owner) public view returns (uint256 balance);
}
contract ethMultiSig {
    event Sent(address msgSender, address toAddress, address otherSigner, bytes data, uint value, bytes32
hash);
    event Deposited(uint value, address from, bytes data);
    event SetSafeModeActivated(address msgSender);

    address[] public signers;
    bool public safeMode = false;

    uint constant MAX_SEQUENCE_ID_SIZE = 10;
    uint[10] recentSequenceIds;
    //Cheetah Mobile Security//构造函数使用正确
    constructor(address[] memory allowedSigners) public {
        if (allowedSigners.length != 3) { //Cheetah Mobile Security//对数组长度进行判断
            revert();
        }
        signers = allowedSigners;
    }
```

```
//Cheetah Mobile Security//回调函数使用正确
function() external payable {
    if (msg.value > 0) {
    emit Deposited(msg.value, msg.sender, msg.data);
  }
}

//Cheetah Mobile Security//判断一个地址是否注册为多签地址，view 修饰符使用正确
function isSigner(address signer) public view returns (bool) {
  for (uint i = 0; i < signers.length; i++) {
    if (signers[i] == signer) {
      return true;
    }
  }
  return false;
}

modifier onlySigner {
  if (!isSigner(msg.sender)) {
    revert();
  }
  _;
}

function sendEth(uint value, uint expireTime, bytes memory data, uint sequenceId, address payable
toAddress, bytes memory signature) public payable onlySigner {
```

```
    bytes32 hash = keccak256(abi.encodePacked("ETHER", toAddress, value, data, expireTime,
sequenceId));
     //Cheetah Mobile Security/// 从签名中恢复出一个多签地址
    address otherSigner = verifySignature(toAddress, hash, signature, expireTime, sequenceId);
    if( !toAddress.send(value)){
        revert();
    }
    emit Sent(msg.sender, toAddress, otherSigner, data, value, hash);
  }

  function sendToken(uint value, uint expireTime, uint sequenceId, address toAddress, address
tokenContractAddress, bytes memory signature) public onlySigner {
    bytes32 hash = keccak256(abi.encodePacked("ERC20", toAddress, value, tokenContractAddress,
expireTime, sequenceId));
    verifySignature(toAddress, hash, signature, expireTime, sequenceId);
    ERC20Interface instance = ERC20Interface(tokenContractAddress);
    if (!instance.transfer(toAddress, value)) {
        revert();
    }
  }

  function verifySignature(
      address toAddress,
      bytes32 hash,
      bytes memory signature,
      uint expireTime,
```

```
        uint sequenceId
    ) private returns (address) {
        address otherSigner = recoverFromSignature(hash, signature);
        // Verify if we are in safe mode. In safe mode, the wallet can only send to signers
        if (safeMode && !isSigner(toAddress)) {
            revert();
        }
        //Cheetah Mobile Security//判断交易有效时间和当前区块时间的大小
        if (expireTime < block.timestamp) {
            revert();
        }
        //Cheetah Mobile Security//更新序列号列表
        insertSequenceId(sequenceId);
        if (!isSigner(otherSigner)) {
            revert();
        }
        //Cheetah Mobile Security//检查 otherSigner 和 msg.address 的重合性
        if (otherSigner == msg.sender) {
            revert();
        }
        return otherSigner;
    }


    //Cheetah Mobile Security//修改器使用正确
    function setSafeMode() public onlySigner {
        safeMode = true;
```

```
    emit SetSafeModeActivated(msg.sender);
}

function recoverFromSignature(bytes32 hash, bytes memory signature) private pure returns (address) {
    //Cheetah Mobile Security//判断签名长度是否合法
    if (signature.length != 65) {
        revert();
    }
    bytes32 r;
    bytes32 s;
    uint8 v;
    //Cheetah Mobile Security//内联汇编使用正确
    assembly {
        r := mload(add(signature, 32))
        s := mload(add(signature, 64))
        v := and(mload(add(signature, 65)), 255)
    }
    if (v < 27) {
        v += 27;
    }
    return ecrecover(hash, v, r, s);
}

 //Cheetah Mobile Security//修改器使用正确
function insertSequenceId(uint sequenceId) private onlySigner {
    uint minIndex = 0;  //Cheetah Mobile Security// 标记最小序列号值的索引
```

```
    //Cheetah Mobile Security//该 for 循环找到最小值的索引
    for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
            if (recentSequenceIds[i] == sequenceId) {
          revert();
        }
        if (recentSequenceIds[i] < recentSequenceIds[minIndex]) {
          minIndex = i;
        }
    }
    //Cheetah Mobile Security//对传入参数的范围进行判断
    if (sequenceId < recentSequenceIds[minIndex]) {
        revert();
    }
     //Cheetah Mobile Security//对传入参数的范围进行判断
    if (sequenceId > (recentSequenceIds[minIndex] + 10000)) {
            revert();
    }
     //Cheetah Mobile Security//更新最小序列号
    recentSequenceIds[minIndex] = sequenceId;
}
//Cheetah Mobile Security//向外暴露接口获取下一个序列号值
function getNextSequenceId() public view returns (uint) {
    uint maxSequenceId = 0;
    for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
        if (recentSequenceIds[i] > maxSequenceId) {
            maxSequenceId = recentSequenceIds[i];
```

```
        }
      }
      return maxSequenceId + 1;
    }
}
```

2、 mncontract.sol

```
pragma solidity ^0.5.4; //Cheetah Mobile Security//指定编译器版本，符合编码规范

contract ERC20Interface {
    function transfer(address _to, uint256 _value) public returns (bool success);
    function balanceOf(address _owner) public view returns (uint256 balance);
}

contract ethMultiSig {
    event Sent(address msgSender, address toAddress, address[] otherSigners, bytes data, uint value, bytes32
hash);
    event Deposited(uint value, address from, bytes data);
    event SetSafeModeActivated(address msgSender);

    address[] public signers;
    address[] public senders;
    uint countSigners;
    uint minNeedSigners; //Cheetah Mobile Security//指定多签最小门限
    bool public safeMode = false;
```

```
uint constant MAX_SEQUENCE_ID_SIZE = 10;

uint[10] recentSequenceIds;

//Cheetah Mobile Security//构造器使用正确

constructor(address[] memory allowedSigners,address[] memory extraSenders,uint m, uint n) public {
  if (allowedSigners.length != n) {
    revert();
  }
  signers = allowedSigners;
  for (uint i = 0; i < signers.length; i++) {
      senders.push(signers[i]);
  }
  for (uint i = 0; i < extraSenders.length; i++) {
      senders.push(extraSenders[i]);
  }
  countSigners = n;
  minNeedSigners = m;
}

//Cheetah Mobile Security//回调函数使用正确

function() external payable {
  if (msg.value > 0) {
    emit Deposited(msg.value, msg.sender, msg.data);
  }
}

//Cheetah Mobile Security//view 修饰符使用正确

function isSigner(address signer) public view returns (bool) {
  for (uint i = 0; i < signers.length; i++) {
```

```
        if (signers[i] == signer) {
            return true;
        }
    }
    return false;
}

function isSender(address sender) public view returns (bool) {
    for (uint i = 0; i < senders.length; i++) {
        if (senders[i] == sender) {
            return true;
        }
    }
    return false;
}

modifier onlySender {
    if (!isSender(msg.sender)) {
        revert();
    }
    _;
}

function sendEth(uint value, uint expireTime, bytes memory data, uint sequenceId, address payable
toAddress, bytes memory signatures) public payable onlySender {
```

```
    bytes32 hash = keccak256(abi.encodePacked("ETHER", toAddress, value, data, expireTime,
sequenceId));
    address [] memory otherSigners = verifySignature(toAddress, hash, signatures, expireTime, sequenceId);
    if( !toAddress.send(value)){
        revert();
    }
    emit Sent(msg.sender, toAddress, otherSigners, data, value, hash);
  }

  function sendToken(uint value, uint expireTime, uint sequenceId, address toAddress, address
tokenContractAddress, bytes memory signatures) public onlySender {
    bytes32 hash = keccak256(abi.encodePacked("ERC20", toAddress, value, tokenContractAddress,
expireTime, sequenceId));
    verifySignature(toAddress, hash, signatures, expireTime, sequenceId);
    ERC20Interface instance = ERC20Interface(tokenContractAddress);
    if (!instance.transfer(toAddress, value)) {
        revert();
    }
  }

  // bytes 动态长度字节数据
  function verifySignature(address toAddress, bytes32 hash, bytes memory signatures, uint expireTime, uint
sequenceId) private returns (address [] memory) {
    if (safeMode && !isSigner(toAddress)) {
        revert(); //代码缩进
    }
```

```
if (expireTime < block.timestamp) {
    revert();  //代码缩进
}
//Cheetah Mobile Security//更新序列号
insertSequenceId(sequenceId);
uint signatureLength = 65;
//Cheetah Mobile Security//对传入参数的长度进行检查
if( signatures.length != minNeedSigners * signatureLength ){
    revert();
}
address [] memory otherSigners = new address[](minNeedSigners);
for (uint i = 0; i < minNeedSigners ; i++) {
  bytes memory curSignatures = new bytes(signatureLength);
  //Cheetah Mobile Security//内层 for 循环获取单个签名
  for( uint j = 0 ; j < 65   ; j++){
      curSignatures[j] = signatures[ i*signatureLength + j];
  }
  //Cheetah Mobile Security//根据单个签名恢复出多签地址
  address otherSigner = recoverFromSignature(hash, curSignatures);
 //Cheetah Mobile Security//对恢复出的地址作多签地址有效性检查
  if (!isSigner(otherSigner)) {
    revert();
  }
  //Cheetah Mobile Security//判断 otherSigner 是否在 otherSigners 中
  if( isInArray(otherSigner, otherSigners)){
      revert();
```

```
        }
        //Cheetah Mobile Security//添加签名地址
        otherSigners[i] = otherSigner;
    }
    return otherSigners;
  }

  function setSafeMode() public onlySender {
    safeMode = true;
    emit SetSafeModeActivated(msg.sender);
  }

  function recoverFromSignature(bytes32 operationHash, bytes memory signature) private pure returns
(address) {
    //Cheetah Mobile Security//检查单个签名长度
    if (signature.length != 65) {
      revert();
    }
    bytes32 r;
    bytes32 s;
    uint8 v;
    //Cheetah Mobile Security//内联汇编使用正确
    assembly {
      r := mload(add(signature, 32))
      s := mload(add(signature, 64))
      v := and(mload(add(signature, 65)), 255)
```

```
    }
    if (v < 27) {
        v += 27;
    }
    return ecrecover(operationHash, v, r, s);
}

function insertSequenceId(uint sequenceId) private onlySender {
    uint minIndex = 0;
    for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
        if (recentSequenceIds[i] == sequenceId) {
            revert();
        }
        if (recentSequenceIds[i] < recentSequenceIds[minIndex]) {
            minIndex = i;
        }
    }
    if (sequenceId < recentSequenceIds[minIndex]) {
        revert();
    }
    if (sequenceId > (recentSequenceIds[minIndex] + 10000)) {
      revert();
    }
    recentSequenceIds[minIndex] = sequenceId;
}
//Cheetah Mobile Security//暴露的获取下一个序列号的接口
```

```
    function getNextSequenceId() public view returns (uint) {
        uint maxSequenceId = 0;
        for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
            if (recentSequenceIds[i] > maxSequenceId) {
                maxSequenceId = recentSequenceIds[i];
            }
        }
        return maxSequenceId + 1;
    }

    function isInArray(address item, address [] memory items) private pure returns (bool) {
        for (uint i = 0; i < items.length; i++) {
            if (items[i] == item) {
                return true;
            }
        }
        return false;
    }
}
```

猎豹区块链安全中心

audit@cmcm.com