MYKEY

智能合约安全审计报告

2019-09-25

概要

在本报告中，我们对 MYKEY 项目的智能合约代码进行安全审计。我们的任务是发现和指出项目里智能合约代码中的安全问题。

声明

慢雾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，慢雾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慢雾提供的文件和资料(简称“已提供资料”)。慢雾假设：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，慢雾对由此而导致的损失和不利影响不承担任何责任。

总结

在本报告中，我们对 MYKEY 项目的智能合约代码进行安全审计。审计没有发现严重、高危的问题，发现了一些**中危、低危程度的安全问题。**经双方沟通反馈，问题均已修复。

项目概述

项目描述

我们审计了 MYKEY 的智能合约代码，如下是相关的文件信息:

审计过程历史版本:

mykey-eth-contracts-slowmist-review-0718.zip:

ca252615e98e1c7d0b63be59fe2899a9e6b9ca298a36f8322fe96464e8271b8a

mykey-eth-contracts-slowmist-review-0801.zip:

be1696a89c765481823fedc37c629a342682211a2deba4f3ab549f671952d58a

mykey-eth-contracts-slowmist-review-0819.zip:

9f717a25585d50f713a16278233cad6265aa3dec8a5e6d30ad8eecfcf20632b7

mykey-eth-contracts-slowmist-review-0821.zip:

6d793d98636199afc20ce82b4300594c00cc2a1c11318e0b4509f0ad1e2a9b19

mykey-eth-contracts-slowmist-review-0828.zip

8cd6a882f0c0eba4f3c30a361a84ef04fbb4435669c1b9b520672118d0bca3f6

mykey-eth-contracts-slowmist-review-0909.zip

73cfe477b993db163b01ab676b4c9866ee29627786a5e3142f9d3d24ca9076d1

mykey-eth-contracts-slowmist-review-0920.zip

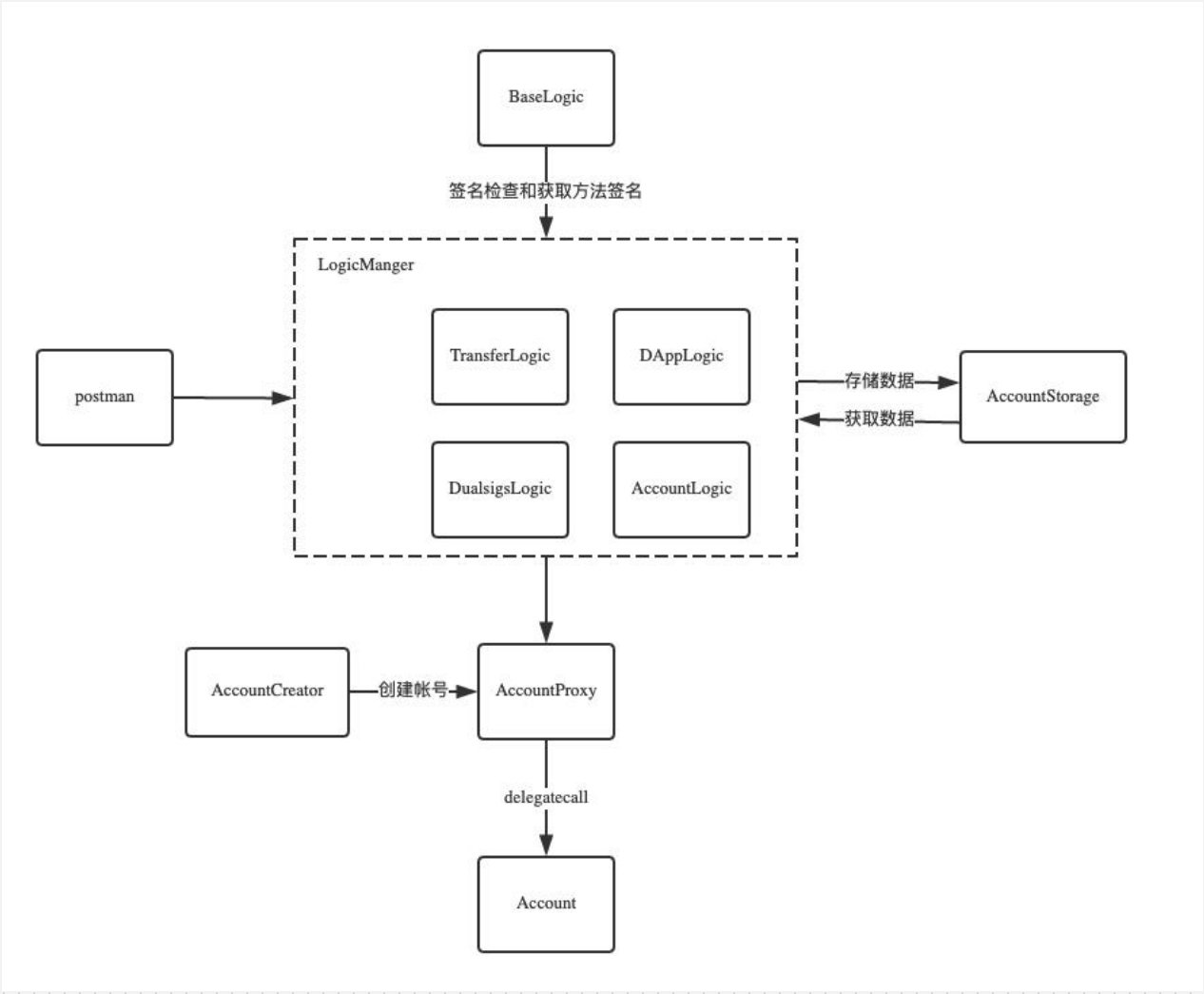
484bde28645b9acd920f0e09c313191eb28ee8fe8c0cbf27cb187c71c8171333

项目结构

```
./contracts
contracts
├── Account.sol
├── AccountCreator.sol
├── AccountProxy.sol
├── AccountStorage.sol
├── LogicManager.sol
├── Migrations.sol
├── logics
│   ├── AccountLogic.sol
│   ├── DappLogic.sol
│   ├── DualsigsLogic.sol
│   ├── TransferLogic.sol
│   └── base
│       ├── AccountBaseLogic.sol
│       └── BaseLogic.sol
├── testUtils
│   ├── MyNft.sol
│   └── MyToken.sol
└── utils
    ├── Owned.sol
    └── SafeMath.sol
```

合约架构

MYKEY 采用创建合约的方式创建一个账号。AccountCreator 合约负责账号的创建，每一个账号在 MYKEY 体系中都是一个合约。用户账号合约本身存储少量数据，并通过 delegatecall Account 合约来加载合约逻辑，减少用户的部署成本。用户数据存储在 AccountStorage 合约中。MYKEY 体系中目前有四个功能模块，分别为 TransferLogic,、DApp Logic、DualsigsLogic、AccountLogic。用于处理转账、DApp 交互、多签及用户账号信息更改，由 LogicManger 合约对模块进行统一的管理。任何拥有用户签名的 ETH 地址（postman），都可以代替用户发起操作。合约总体架构如下图所示:



审计方法

我们的智能合约安全审计流程包含两个步骤:

- 使用开源或内部自动化分析的工具对合约代码中常见的安全漏洞进行扫描和测试。
- 人工审计代码的安全问题，通过人工分析合约代码，查找代码中潜在的安全问题。

如下是合约代码审计过程中我们会重点审查的常见漏洞列表:

- 重入攻击
- 重放攻击
- 重排攻击
- 数据存储问题
- 短地址攻击
- 拒绝服务攻击
- 交易顺序依赖
- 条件竞争攻击
- 权限控制攻击
- 整数上溢/下溢攻击
- 时间戳依赖攻击
- Gas 使用，Gas 限制和循环
- 冗余的回调函数
- 不安全的接口使用
- 函数状态变量的显式可见性
- 业务逻辑缺陷
- 未声明的存储指针
- 算术精度误差
- tx.origin 身份验证
- 假充值漏洞
- Event 事件安全
- 编译器版本问题
- call 调用安全

审计结果

严重漏洞

严重漏洞会对智能合约的安全造成重大影响，强烈建议修复严重漏洞。

经过审计该项目未发现严重漏洞。

高危漏洞

高危漏洞会影响智能合约的正常运行，强烈建议修复高危漏洞。

经过审计该项目未发现高危漏洞。

中危漏洞

中危漏洞会影响智能合约的运行，建议修复中危漏洞。

1、Gas 恶意消耗问题

由于 TransferLogic 和 DAppLogic 合约逻辑流程上最终调用的是 AccountProxy 合约中的 fallback 函数，最终使用 assembly 对外部合约进行调用的时候合约没有对 gas 进行限制， 导致攻击者可能通过构建恶意合约消耗和盗窃用户的 gas 或官方自己的资产。

```
function() external payable {

        if(msg.data.length == 0 && msg.value > 0) {
            emit Received(msg.value, msg.sender, msg.data);
        }
        else {
            // solium-disable-next-line security/no-inline-assembly
            assembly {
                let target := sload(0)
                calldatacopy(0, 0, calldatasize())
                let result := delegatecall(gas, target, 0, calldatasize(), 0, 0)
                returndatacopy(0, 0, returndatasize())
```

```
                switch result
                case 0 {revert(0, returndatasize())}
                default {return (0, returndatasize())}
            }
        }
    }
}
```

修复情况：将使用服务端对 gas 进行限制和监控。

2、转账模块功能缺失

TransferLogic 合约中由于在入口函数 enter()中对 data 数据进行校验，其操作账户必须为用户（AccountProxy 合约）本身，导致进行 ERC721 和 ERC20 代币转移的时候无法通过 approve-transferFrom 的方式进行转移。

```
    function enter(address _account, bytes calldata _signature, bytes calldata _data, uint256 _nonce) external {
        checkData(_account, _data);
        checkAndUpdateNonce(_account, _nonce, TranferKeyIndex);

        address assetKey = accountStorage.getKeyData(_account, TranferKeyIndex);
        bytes32 signHash = getSignHash(_account, _data, _nonce);
        verifySig(assetKey, _signature, signHash);

        // solium-disable-next-line security/no-low-level-calls
        (bool success,) = address(this).call(_data);
        require(success, "calling self failed");
    }
```

修复情况：增加 transferApprovedErc20 和 transferApprovedNft 两个方法（v0819 修复）。

低危漏洞

低危漏洞可能会影响未来版本代码中智能合约的操作，建议项目方自行评估和考虑这些问题是否需要修复。

1、事件记录错误

在 transferLogic.sol 文件中的 transferERC20 方法中，由于 token 参数是调用者可以自己任意填的，就算这个地址不存在，Account 合约的 invoke 调用也不会失败，假如恶意用户的 token 地址填的是 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE 的话，就会造成事件记录错误。

```
function transferEth(address payable _from, address payable _to, uint256 _amount) external onlySelf {
        BaseAccount(_from).invoke(_to, _amount, "");
        emit Transfer(_from, _to, ETH_TOKEN, _amount);
    }

function transferErc20(address payable _from, address payable _to, address _token, uint256 _amount) external onlySelf
{
        bytes memory methodData = abi.encodeWithSignature("transfer(address,uint256)", _to, _amount);
        BaseAccount(_from).invoke(_token, 0, methodData);
        emit Transfer(_from, _to, _token, _amount);
    }
```

修复情况：简化事件设计，改用每个 logic 合约都在 enter 入口处进行统一的事件处理（v0801 修复）。

2、事件声明混淆

BaseLogic 合约中的 initAccount 函数与 TransferLogic 合约中 initAccount 函数声明的事件一致，可能造成混淆。

BaseLogic:

```
function initAccount(BaseAccount _account) external onlyAccount(_account){
       emit LogicInitialised(address(_account));
    }
```

TransferLogic：

```
function initAccount(BaseAccount _account) external onlyAccount(_account){
       _account.enableStaticCall(address(this), ERC721_RECEIVED);
```

```
        emit LogicInitialised(address(_account));
}
```

修复情况：TransferLogic 合约在 enter 入口采用统一的事件处理，与 BaseLogic 合约中的事件区分（v0801 修复）。

3、没有对传入的用户公钥对和紧急联系人帐号数量做限制

使用 Account 合约初始化帐号的时候，没有对传入的用户公钥对和紧急联系人帐号数量做限制，用户可以上传任意数量的公钥，导致某些功能可能无法正常使用。如通过 getKeyIndex 获取签名公钥的函数。

```
function getKeyIndex(bytes memory _data) internal pure returns (uint256) {
                uint256 index;
                bytes4 methodId = getMethodId(_data);
                if (methodId == bytes4(keccak256("addOperationKey(address,address,uint256)"))) {
                        index = 2; //adding key
                } else if (methodId == bytes4(keccak256("proposeByBackup(address,address,bytes)"))) {
                        index = 4; //assist key
                } else if (methodId == bytes4(keccak256("approveProposal(address,address,bytes32)"))) {
                        index = 4; //assist key
                } else {
                        index = 0; //admin key
                }
                return index;
        }
```

修复情况：使用服务端控制公钥数量，在进行用户私钥创建的时候会一次性传入至少 5 个私钥。

4、过期 delay 操作可被取消

AccountLogic 中，在调用 cancelDealy 函数的时候没有对 delay 时间进行判断，导致超过 delay 时间仍然可以被取消。

```
function cancelDelayed(address payable _account, bytes32 _id) public onlySelf {
```

```
                accountStorage.clearDelayedData(_account, _id);
                emit CancelDelayedDone(_account, _id);
        }
```

修复情况：经与项目方确认，cancelDelay 方法不检查到期时间(原因：如果延时到期后无法再 cancel，某些临界情况下 MYKEY 后端可能无法保证多链一致性，如 EOS 链上 cancel changeAdminKey 成功，ETH 链因为入块较慢而没有及时 cancel 的情况)。

5、未对 OperationKeys 做一致性校验

changeAllOperationKeys 未检验 keys 是否一致，建议增加校验。

```
function changeAllOperationKeys(address payable _account, address[] calldata _pks) external onlySelf {
                require(_pks.length == MAX_DEFINED_OPR_KEY_INDEX, "invalid numbers of keys");
                bytes4 actionId = getActionId("changeAllOperationKeys(address,address[])");
                require(accountStorage.getDelayDataHash(_account, actionId) == 0, "delay data already
exists");
                for (uint256 i = 0; i < MAX_DEFINED_OPR_KEY_INDEX; i++) {
                        address pk = _pks[i];
                        require(pk != address(0), "0x0 is invalid");
                }
                bytes32 hash = keccak256(abi.encodePacked('changeAllOperationKeys', _account, _pks));
                accountStorage.setDelayData(_account, actionId, hash, now +
getDelayTime(TYPE_CHANGE_OPERATION_KEY));
        }
```

修复情况: 经与项目方沟通，确定合约层不做 keys 重复性检查。理论上客户端不会生成重复的 key，服务端也会做检查。

6、未对 Backupkey 作最低数量限制

BackupKey 数量没有最低限制，导致 Backup 账户可能在用户不知情的情况下做恶，更改主账户的公钥，盗窃用户资产，在单 backup 账户的情况下可以直接发起 proposal 后绕过 approveProposal 然后再调用

executeProposal 后等待 30 天即可更改用户主公钥(Adminkey)。

攻击流程：

（1）备份账户发起提案。

```
function proposeByBackup(address _backup, address payable _client, bytes memory _functionData) public onlySelf {
                require(getMethodId(_functionData) ==
bytes4(keccak256("changeAdminKeyByBackup(address,address)")), "invalid proposal by backup");
                checkRelation(_client, _backup);
                bytes32 functionHash = keccak256(_functionData);
                accountStorage.setProposalData(_client, functionHash, _backup);
                emit ProposeByBackupDone(_backup, _client, _functionData);
        }饭店吃饭
```

(2）在 BackupKey 的情况下，备份账户直接发起 executeProposal 操作，等待 30 天后系统自动 trigger 后更改账户主公钥。

```
function changeAdminKeyByBackup(address payable _account, address _pkNew) public onlySelf {
                address pk = accountStorage.getKeyData(_account, 0);
                require(pk != _pkNew, "identical admin key exists");
                require(_pkNew != address(0), "0x0 is invalid");
                bytes32 id = keccak256(abi.encodePacked('changeAdminKeyByBackup', _account, _pkNew));
                accountStorage.setDelayedData(_account, id, now +
getDelayTime(TYPE_CHANGE_ADMIN_KEY_BY_BACKUP));
                emit ChangeAdminKeyByBackupSubmitted(_account, _pkNew);
        }
        function triggerChangeAdminKeyByBackup(address payable _account, address _pkNew) public {
                bytes32 id = keccak256(abi.encodePacked('changeAdminKeyByBackup', _account, _pkNew));
                uint256 due = accountStorage.getDelayedData(_account, id);
                require(due > 0, "delayed data not found");
                require(due < now, "too early to trigger changeAdminKeyByBackup");
                accountStorage.setKeyData(_account, 0, _pkNew);
                accountStorage.clearDelayedData(_account, id);
                emit ChangeAdminKeyByBackupTriggered(_account, _pkNew);
        }
```

修复情况：经与项目方确认，目前第一个默认紧急联系人为 MYKEY 官方，任何账户相关操作都会通知用户。

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account