MYKEY

Smart Contract Security Audit

2020-02-27

Abstract

This report provides a comprehensive view of the security audit results for the smart contract code of the MYKEY project. The task of SlowMist is to review and point out security issues in the audited smart contract code.

Disclaimer

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these. For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of the smart contract, and is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of this report (referred to as "the provided information"). If the provided information is missing, tampered, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom.

Summary

In this report, SlowMist audited the smart contract code for the MYKEY project. The audit results showed that no issues with critical severity or high severity were found. However, some issues with medium severity and low severity were discovered. After communication and feedback from both parties, the issues have been fixed.

Project Overview

Description

The following are the details of the smart contract code of the audited project (MYKEY):

The new audit content:

Github commit:

https://github.com/mykeylab/keyid-eth-contracts/commit/c7a62bb2d868c413dbca7971a7cfad7d53a8ba91

Contract address & contract hash（SHA256）:

Contract Address：0x847f5AbbA6A36c727eCfF76784eE3648BA868808

Hash：9dbc0e717dfbac8cb46ed2883d1dd91b23aab7ff87473d0e35947109d5a0840c

The new file hash（SHA256）:

DappLogic.sol:

9dbc0e717dfbac8cb46ed2883d1dd91b23aab7ff87473d0e35947109d5a0840c

The contract address:

https://etherscan.io/address/0x847f5abba6a36c727ecff76784ee3648ba868808#code

This new audit has been passed the consistency check on Github and Etherscan

The history audit version

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

mykey-eth-slowmist-review-1011.zip

e594addf0367a89eeee9aebc6a795796cb1dac2f896564fc5096addba46a4fd7

mykey-eth-slowmist-review-1115.zip

c90a1e3140d22fb36b6a81a9d38ff98314edb16737df2caa4c652df0731aa0e2

mykey-eth-slowmist-review-1120.zip

91438cd1bcfabdd7d887b1c96153c214c898d539aa4f1b671b5784f9255a505d

mykey-eth-slowmist-review-1129.zip

827a28ec95ac5d0d3d4ccd42bff8da3344f2aaa1868eacac3f062c254fcb3541

mykey-eth-slowmist-review-1227.zip

2d0636976229be20f93a344f9e208a73395600fd4c0630f16b5cb31eeab74be3

mykey-eth-slowmist-review-20200219.zip

f3a5326b58c981d332e9e58280c530e5b113064ece39a9efa7bd659a408fa1e8

The contract address of the project:

LogicManager:

https://etherscan.io/address/0xDF8aC96BC9198c610285b3d1B29de09621B04528#code

AccountLogic:

https://etherscan.io/address/0x52dab11c6029862ebf1e65a4d5c30641f5fbd957#code

TransferLogic:

https://etherscan.io/address/0x1C2349ACBb7f83d07577692c75B6D7654899BF10#code

DualsigsLogic:

https://etherscan.io/address/0x039aA54fEbe98AaaDb91aE2b1Db7aA00a82F8571#code

DappLogic:

https://etherscan.io/address/0x847f5abba6a36c727ecff76784ee3648ba868808#code

AccountCreator:

https://etherscan.io/address/0x185479FB2cAEcbA11227db4186046496D6230243#code

Account:

https://etherscan.io/address/0xEf004D954999EB9162aeB3989279eFf2161D5095#code

AccountStorage :

https://etherscan.io/address/0xADc92d1fD878580579716d944eF3460E241604b7#code

All the contracts above have passed the consistency check on Github and Etherscan

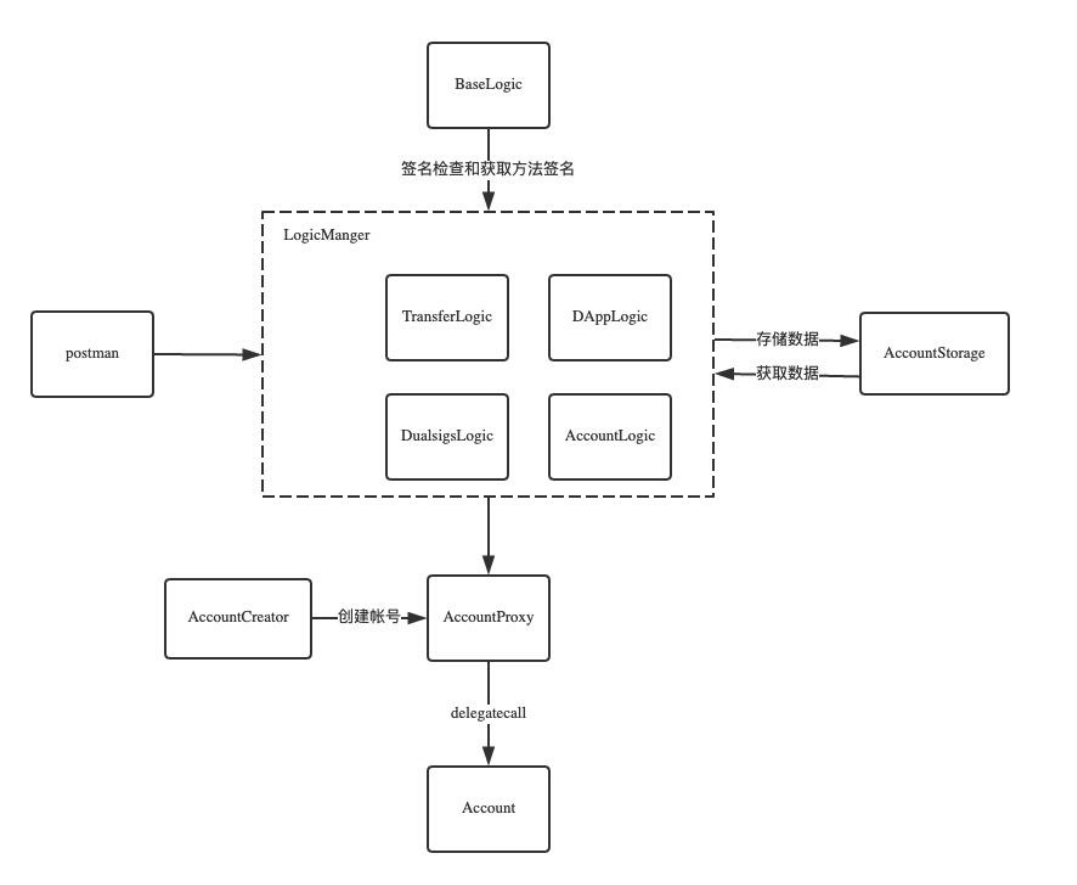
Project Structure

The project includes the following smart contract files:

```
./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
    ├── MultiOwned.sol
    ├── Owned.sol
    ├── RLPReader.sol
    └── SafeMath.sol
```

Contracts Structure

MYKEY adopts the strategy of creating a contract for each account. AccountCreator contract is responsible for the account creation, and each account in MYKEY system is a contract. The account contract only stores a few data and loads the contract logic by delegatecall Account contract to reduce the user deployment cost. The account data is stored in AccountStorage contract. The MYKEY system has four modules now, which are TransferLogic, DApp Logic, DualsigsLogic, AccountLogic and used for the transfer, interact with DApp, multi-sign and account information modification. All these modules are managed by the LogicManger contract. Every ETH address (postman) who has the account signature can initiate an operation for the account. The overall structure of the contract is shown below:



Audit Methodology

The security audit process for smart contracts consists of the following two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.

- Manually audit the security of the code. Discover the potential security issues in the code by manually analyzing the contract code.

The following is a list of common vulnerabilities that will be highlighted during the contract code audit:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Data Storage issue
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack
- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Business Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Event Security
- Compiler version issues
- Call function Security

Audit Result

Critical Vulnerabilities

Critical severity issues can have a major impact on the security of smart contracts, and it is highly recommended to fix critical severity vulnerability.

The audit has shown no critical severity vulnerability.

High Vulnerabilities

High severity issues can affect the normal operation of smart contracts, and it is highly recommended to fix high severity vulnerability.

The audit has shown no high severity vulnerability.

Medium Vulnerabilities

Medium severity issues can affect the operation of a smart contract, and it is recommended to fix medium severity vulnerability.

1. Gas malicious consumption

The TransferLogic and the DAppLogic will finally call the fallback function of the AccountProxy contract, and eventually invoke the external contract through assembly, but in this flow, the contract does not limit the gas and the attacker can deploy a malicious contract to consume and steal the gas of the account and the asset of the project side.

```
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())}
            }
        }
    }
}
```

Fix status: Will limit and monitor the gas on the server-side.

2. Function Missing in transfer module:

Because TransferLogic contract must check the data in enter() function, the account that operates must be user(The AccountProxy contract) itself, which makes user can not transfer ERC721 and ERC20 by approve-transferFrom.

```
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");
```

```
        }
```

Fix status: Add transferApproveErc20 function and transferApprovedNft function. (fixed in v0819)

Low Vulnerabilities

Low severity issues can affect smart contracts operation in future versions of code. We recommend the project party to evaluate and consider whether these problems need to be fixed.

1. Event log error

In transferERC20 function of the file transferLogic.sol, the token parameter can be filled by the caller arbitrarily, and even if this address does not exist, the invoke call of the Ac count contract will not fail. If the malicious user's token address is filled with 0xEeeeeEe eeEeEeEeEeeEeeeeeeEeeeeeeeeEEeE, the event record error will occur.

```
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);
}
```

Fix status: Simplify event design, use unified logic for event handling at the entry of each logic contract. (fixed in v0801)

2. Confused Event Declaration

The event declared in initAccount of TransferLogic and BasicLogic is the same and may lead to confusion.

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));
}
```

Fix status: The TransferLogic contract uses unified event processing in the enter() to distinguish it from events in the BaseLogic contract.

3. No restrictions on the number of user public key pairs and backup accounts

When using the Account contract to initialize an account, there is no limit on the number of pass-in user public key pairs and backup accounts. Users can upload any number of public

keys, resulting in some functions may not work properly. Such as the function to get the public key of the signature by getKeyIndex.

```
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;
        }
```

Fix status: Control the number of public keys on the server-side. When creating the user's private key, at least 5 private keys will be passed in at one time.

4. Expired delay operation can be canceled

In Account Logic, when the cancelDealy function is called, the delay time is not judged, so that the delayed operation can still be canceled.

```
function cancelDelayed(address payable _account, bytes32 _id) public onlySelf {
                accountStorage.clearDelayedData(_account, _id);
                emit CancelDelayedDone(_account, _id);
        }
```

Fix status: After confirming with the project party, cancelDelay method does not check the expiration time. (reason: if the delayed operation cannot be canceled after the delay time expires, the MYKEY backend may not guarantee multi-chain consistency in some critical cases, such as cancel on the EOS chain changeAdminKey succeeded, and the ETH chain did not cancel in time because the block was inserted slowly).

5. No consistency check on OperationKeys

The changeAllOperationKeys function does not check whether the keys are the same, it is recommended to add a check.

```
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));
                        }
```

Fix status: After communicating with the project party, it is determined that the contract layer does not take the repeated keys checking. In theory, the client will not generate duplicate keys, and the server will check.

6. No minimum restriction number of Backupkeys

There is no minimum restriction number of BackupKeys, causing the Backup account to do evil without the user's knowledge, change the public key of the main account, and steal user assets. In the case of a single backup account, you can directly initiate a proposal, bypass approvedProposal, and then call executeProposal. After waiting 30 days, you can change the user's master public key (Adminkey).

Attack flow:

(1) The backup account initiate the proposal.

```
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);
                }
```

(2) In the case of BackupKey, the backup account directly initiates the executeProposal operation. After 30 days, the system automatically triggers and changes the account's admin public key.

```
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);
        }
```

Fix status: After confirming with the project party, the first default backup account is currently MYKEY official, and any account related operations will be notified to the user.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account