智能合约安全审计报告

慢雾安全团队于 2019-07-18 日，收到 Newton 团队对多签钱包项目智能合约安全审计申请。如下为本次智能合约安全审计细节及结果：

文件名称：

MultiSigWallet.sol

MultiSigWalletWithDailyLimit.sol

文件 MD5 值：

MD5 (MultiSigWallet.sol) = 417bdfe0dab52723aa035b2adf957e2a

MD5 (MultiSigWalletWithDailyLimit.sol) = 64ee7322514221c0b0765eb923efb58a

本次审计项及结果：

（其他未知安全漏洞不包含在本次审计责任范围）

序号	审计大类	审计子类	审计结果
1	溢出审计	-	通过
2	条件竞争审计	-	通过
3	权限控制审计	权限漏洞审计	通过
		权限过大审计	通过
4	安全设计审计	Zeppelin 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
		Fallback 函数使用安全	通过
		显现编码安全	通过
		函数返回值安全	通过
		call 调用安全	通过
5	拒绝服务审计	-	通过
6	Gas 优化审计	-	通过
7	设计逻辑审计	-	通过

8	“假充值”漏洞审计	-	通过
9	恶意 Event 事件日志审计	-	通过
10	未初始化的存储指针	-	通过
11	算术精度误差	-	通过

备注：审计意见及建议见代码注释 **//SlowMist//……**

审计结果： **通过**

审计编号：0X001908010001

审计日期：2019 年 08 月 01 日

审计团队：慢雾安全团队

总结：此为多签钱包合约。综合评估合约无溢出，条件竞争风险。

合约源代码如下：

文件一：MultiSigWallet.sol

```
pragma solidity ^0.4.15;


/// @title Multisignature wallet - Allows multiple parties to agree on transactions before execution.
/// @author Stefan George - <stefan.george@consensys.net>
contract MultiSigWallet {

    /*
     *  Events
     */
    event Confirmation(address indexed sender, uint indexed transactionId);
    event Revocation(address indexed sender, uint indexed transactionId);
    event Submission(uint indexed transactionId);
    event Execution(uint indexed transactionId);
```

```
event ExecutionFailure(uint indexed transactionId);
event Deposit(address indexed sender, uint value);
event OwnerAddition(address indexed owner);
event OwnerRemoval(address indexed owner);
event RequirementChange(uint required);

/*
 *  Constants
 */
uint constant public MAX_OWNER_COUNT = 50;

/*
 *  Storage
 */
mapping (uint => Transaction) public transactions;
mapping (uint => mapping (address => bool)) public confirmations;
mapping (address => bool) public isOwner;
address[] public owners;
uint public required;
uint public transactionCount;

struct Transaction {
    address destination;
    uint value;
    bytes data;
    bool executed;
}

/*
 *  Modifiers
 */
modifier onlyWallet() {
    require(msg.sender == address(this));
    _;
}

modifier ownerDoesNotExist(address owner) {
    require(!isOwner[owner]);
    _;
}

modifier ownerExists(address owner) {
```

```
        require(isOwner[owner]);
        _;
    }

    modifier transactionExists(uint transactionId) {
        require(transactions[transactionId].destination != 0);
        _;
    }

    modifier confirmed(uint transactionId, address owner) {
        require(confirmations[transactionId][owner]);
        _;
    }

    modifier notConfirmed(uint transactionId, address owner) {
        require(!confirmations[transactionId][owner]);
        _;
    }

    modifier notExecuted(uint transactionId) {
        require(!transactions[transactionId].executed);
        _;
    }

    modifier notNull(address _address) {
        require(_address != 0);   //SlowMist// 这类检查很好，避免操作失误导致意外。
        _;
    }

    modifier validRequirement(uint ownerCount, uint _required) {
        require(ownerCount <= MAX_OWNER_COUNT
            && _required <= ownerCount
            && _required != 0
            && ownerCount != 0);
        _;
    }

    /// @dev Fallback function allows to deposit ether.
    function()
        payable
    {
```

```
        if (msg.value > 0)
            Deposit(msg.sender, msg.value);
    }

    /*
     * Public functions
     */
    /// @dev Contract constructor sets initial owners and required number of confirmations.
    /// @param _owners List of initial owners.
    /// @param _required Number of required confirmations.
    function MultiSigWallet(address[] _owners, uint _required)
        public
        validRequirement(_owners.length, _required)
    {
        for (uint i=0; i<_owners.length; i++) {
            require(!isOwner[_owners[i]] && _owners[i] != 0); //SlowMist// 这类检查很好，避免操作失误导致
意外。
            isOwner[_owners[i]] = true;
        }
        owners = _owners;
        required = _required;
    }

    /// @dev Allows to add a new owner. Transaction has to be sent by wallet.
    /// @param owner Address of new owner.
    function addOwner(address owner)
        public
        onlyWallet
        ownerDoesNotExist(owner)
        notNull(owner)
        validRequirement(owners.length + 1, required)
    {
        isOwner[owner] = true;
        owners.push(owner);
        OwnerAddition(owner);
    }

    /// @dev Allows to remove an owner. Transaction has to be sent by wallet.
    /// @param owner Address of owner.
    function removeOwner(address owner)
```

```
        public
        onlyWallet
        ownerExists(owner)
    {
        isOwner[owner] = false;
        for (uint i=0; i<owners.length - 1; i++)
            if (owners[i] == owner) {
                owners[i] = owners[owners.length - 1];
                break;
            }
        owners.length -= 1;
        if (required > owners.length)
            changeRequirement(owners.length);
        OwnerRemoval(owner);
    }

    /// @dev Allows to replace an owner with a new owner. Transaction has to be sent by wallet.
    /// @param owner Address of owner to be replaced.
    /// @param newOwner Address of new owner.
    function replaceOwner(address owner, address newOwner)
        public
        onlyWallet
        ownerExists(owner)
        ownerDoesNotExist(newOwner)
    {
        for (uint i=0; i<owners.length; i++)
            if (owners[i] == owner) {
                owners[i] = newOwner;
                break;
            }
        isOwner[owner] = false;
        isOwner[newOwner] = true;
        OwnerRemoval(owner);
        OwnerAddition(newOwner);
    }

    /// @dev Allows to change the number of required confirmations. Transaction has to be sent by wallet.
    /// @param _required Number of required confirmations.
    function changeRequirement(uint _required)
        public
        onlyWallet
        validRequirement(owners.length, _required)
```

```
{
    required = _required;
    RequirementChange(_required);
}

/// @dev Allows an owner to submit and confirm a transaction.
/// @param destination Transaction target address.
/// @param value Transaction ether value.
/// @param data Transaction data payload.
/// @return Returns transaction ID.
function submitTransaction(address destination, uint value, bytes data)
    public
    returns (uint transactionId)
{
    transactionId = addTransaction(destination, value, data);
    confirmTransaction(transactionId);
}

/// @dev Allows an owner to confirm a transaction.
/// @param transactionId Transaction ID.
function confirmTransaction(uint transactionId)
    public
    ownerExists(msg.sender)
    transactionExists(transactionId)
    notConfirmed(transactionId, msg.sender)
{
    confirmations[transactionId][msg.sender] = true;
    Confirmation(msg.sender, transactionId);
    executeTransaction(transactionId);
}

/// @dev Allows an owner to revoke a confirmation for a transaction.
/// @param transactionId Transaction ID.
function revokeConfirmation(uint transactionId)
    public
    ownerExists(msg.sender)
    confirmed(transactionId, msg.sender)
    notExecuted(transactionId)
{
    confirmations[transactionId][msg.sender] = false;
    Revocation(msg.sender, transactionId);
}
```

```
    /// @dev Allows anyone to execute a confirmed transaction.
    /// @param transactionId Transaction ID.
    function executeTransaction(uint transactionId)
        public
        ownerExists(msg.sender)
        confirmed(transactionId, msg.sender)
        notExecuted(transactionId)
    {
        if (isConfirmed(transactionId)) {
            Transaction storage txn = transactions[transactionId];
            txn.executed = true;
            if (external_call(txn.destination, txn.value, txn.data.length, txn.data))
                Execution(transactionId);
            else {
                ExecutionFailure(transactionId);
                txn.executed = false;
            }
        }
    }

    // call has been separated into its own function in order to take advantage
    // of the Solidity's code generator to produce a loop that copies tx.data into memory.
    function external_call(address destination, uint value, uint dataLength, bytes data) internal returns (bool) {
        bool result;
        assembly {
            let x := mload(0x40)    // "Allocate" memory for output (0x40 is where "free memory" pointer is stored by convention)
            let d := add(data, 32) // First 32 bytes are the padded length of data, so exclude that
            result := call(
                sub(gas, 34710),    // 34710 is the value that solidity is currently emitting
                                    // It includes callGas (700) + callVeryLow (3, to pay for SUB) + callValueTransferGas (9000) + //SlowMist// 限定 Gas 值，避免恶意消耗，值得称赞的做法
                                    // callNewAccountGas (25000, in case the destination address does not exist and needs creating)
                destination,
                value,
                d,
                dataLength,         // Size of the input (in bytes) - this is what fixes the padding problem
                x,
                0                   // Output is ignored, therefore the output size is zero
```

```
            )
        }
        return result;
    }

    /// @dev Returns the confirmation status of a transaction.
    /// @param transactionId Transaction ID.
    /// @return Confirmation status.
    function isConfirmed(uint transactionId)
        public
        constant
        returns (bool)
    {
        uint count = 0;
        for (uint i=0; i<owners.length; i++) {
            if (confirmations[transactionId][owners[i]])
                count += 1;
            if (count == required)
                return true;
        }
    }

    /*
     * Internal functions
     */
    /// @dev Adds a new transaction to the transaction mapping, if transaction does not exist yet.
    /// @param destination Transaction target address.
    /// @param value Transaction ether value.
    /// @param data Transaction data payload.
    /// @return Returns transaction ID.
    function addTransaction(address destination, uint value, bytes data)
        internal
        notNull(destination)
        returns (uint transactionId)
    {
        transactionId = transactionCount;
        transactions[transactionId] = Transaction({
            destination: destination,
            value: value,
            data: data,
            executed: false
        });
```

```
        transactionCount += 1;
        Submission(transactionId);
    }

    /*
     * Web3 call functions
     */
    /// @dev Returns number of confirmations of a transaction.
    /// @param transactionId Transaction ID.
    /// @return Number of confirmations.
    function getConfirmationCount(uint transactionId)
        public
        constant
        returns (uint count)
    {
        for (uint i=0; i<owners.length; i++)
            if (confirmations[transactionId][owners[i]])
                count += 1;
    }

    /// @dev Returns total number of transactions after filers are applied.
    /// @param pending Include pending transactions.
    /// @param executed Include executed transactions.
    /// @return Total number of transactions after filters are applied.
    function getTransactionCount(bool pending, bool executed)
        public
        constant
        returns (uint count)
    {
        for (uint i=0; i<transactionCount; i++)
            if (    pending && !transactions[i].executed
                || executed && transactions[i].executed)
                count += 1;
    }

    /// @dev Returns list of owners.
    /// @return List of owner addresses.
    function getOwners()
        public
        constant
        returns (address[])
    {
```

```
        return owners;
    }

    /// @dev Returns array with owner addresses, which confirmed transaction.
    /// @param transactionId Transaction ID.
    /// @return Returns array of owner addresses.
    function getConfirmations(uint transactionId)
        public
        constant
        returns (address[] _confirmations)
    {
        address[] memory confirmationsTemp = new address[](owners.length);
        uint count = 0;
        uint i;
        for (i=0; i<owners.length; i++)
            if (confirmations[transactionId][owners[i]]) {
                confirmationsTemp[count] = owners[i];
                count += 1;
            }
        _confirmations = new address[](count);
        for (i=0; i<count; i++)
            _confirmations[i] = confirmationsTemp[i];
    }

    /// @dev Returns list of transaction IDs in defined range.
    /// @param from Index start position of transaction array.
    /// @param to Index end position of transaction array.
    /// @param pending Include pending transactions.
    /// @param executed Include executed transactions.
    /// @return Returns array of transaction IDs.
    function getTransactionIds(uint from, uint to, bool pending, bool executed)
        public
        constant
        returns (uint[] _transactionIds)
    {
        uint[] memory transactionIdsTemp = new uint[](transactionCount);
        uint count = 0;
        uint i;
        for (i=0; i<transactionCount; i++)
            if (    pending && !transactions[i].executed
                || executed && transactions[i].executed)
            {
```

```
                transactionIdsTemp[count] = i;
                count += 1;
            }
        _transactionIds = new uint[](to - from);
        for (i=from; i<to; i++)
            _transactionIds[i - from] = transactionIdsTemp[i];
    }
}
```

文件二：MultiSigWalletWithDailyLimit.sol

```
pragma solidity ^0.4.15;
import "./MultiSigWallet.sol";


/// @title Multisignature wallet with daily limit - Allows an owner to withdraw a daily limit without multisig.
/// @author Stefan George - <stefan.george@consensys.net>
contract MultiSigWalletWithDailyLimit is MultiSigWallet {

    /*
     *  Events
     */
    event DailyLimitChange(uint dailyLimit);

    /*
     *  Storage
     */
    uint public dailyLimit;
    uint public lastDay;
    uint public spentToday;

    /*
     * Public functions
     */
    /// @dev Contract constructor sets initial owners, required number of confirmations and daily withdraw limit.
    /// @param _owners List of initial owners.
    /// @param _required Number of required confirmations.
    /// @param _dailyLimit Amount in wei, which can be withdrawn without confirmations on a daily basis.
    function MultiSigWalletWithDailyLimit(address[] _owners, uint _required, uint _dailyLimit)
        public
```

```
        MultiSigWallet(_owners, _required)
{
        dailyLimit = _dailyLimit;
}

/// @dev Allows to change the daily limit. Transaction has to be sent by wallet.
/// @param _dailyLimit Amount in wei.
function changeDailyLimit(uint _dailyLimit)
        public
        onlyWallet
{
        dailyLimit = _dailyLimit;
        DailyLimitChange(_dailyLimit);
}

/// @dev Allows anyone to execute a confirmed transaction or ether withdraws until daily limit is reached.
/// @param transactionId Transaction ID.
function executeTransaction(uint transactionId)
        public
        ownerExists(msg.sender)
        confirmed(transactionId, msg.sender)
        notExecuted(transactionId)
{
        Transaction storage txn = transactions[transactionId];
        bool _confirmed = isConfirmed(transactionId);
        if (_confirmed || txn.data.length == 0 && isUnderLimit(txn.value)) {
                txn.executed = true;
                if (!_confirmed)
                        spentToday += txn.value;
                if (external_call(txn.destination, txn.value, txn.data.length, txn.data))
                        Execution(transactionId);
                else {
                        ExecutionFailure(transactionId);
                        txn.executed = false;
                        if (!_confirmed)
                                spentToday -= txn.value;
                }
        }
}

/*
 * Internal functions
```

```
     */
    /// @dev Returns if amount is within daily limit and resets spentToday after one day.
    /// @param amount Amount to withdraw.
    /// @return Returns if amount is under daily limit.
    function isUnderLimit(uint amount)
        internal
        returns (bool)
    {
        if (now > lastDay + 24 hours) {
            lastDay = now;
            spentToday = 0;
        }
        if (spentToday + amount > dailyLimit || spentToday + amount < spentToday)
            return false;
        return true;
    }

    /*
     * Web3 call functions
     */
    /// @dev Returns maximum withdraw amount.
    /// @return Returns amount.
    function calcMaxWithdraw()
        public
        constant
        returns (uint)
    {
        if (now > lastDay + 24 hours)
            return dailyLimit;
        if (dailyLimit < spentToday)
            return 0;
        return dailyLimit - spentToday;
    }
}
```

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号