OZYS
Multisig-Wallet
Security Analysis Report

Prepared by

78ResearchLab

RedSpider

Jan 5, 2025

TABLE OF CONTENTS

PROJECT OVERALL

About Project

This project involves deploying and configuring a new Multisig Wallet Contract designed for systems operating on the Silicon and other blockchain networks. According to our client, while the existing Multisig Wallet Contract is simple and reliable, it has become necessary to address certain inefficiencies that have surfaced over prolonged use. This updated version enhances convenience, introduces flexibility, and separates wallets for system operations and development purposes, improving management efficiency.

Target Summary

Name	Multisig-Wallet
Website	
Repository	
Commit	bc47fffb7fef907d3c5af5581c5216b8e1f3eb11
Network	Silicon
Languages	Solidity
Method	Source code auditing
Timeline	Dec 13, 2024 ~ Dec 20, 2024

SCOPE

The audit will focus on reviewing the provided `CommonMultiSigWallet.sol` file, which serves as the sole implementation of the updated Multisig Wallet Contract. Key areas of review include:

- Evaluating the single-file Multisig Wallet Contract for security, reliability, and functionality.
- Assessing the newly added features, such as `confirmTransactionByRange` and improved wallet address management.
- Verifying updates to access control policies and their practical application.

Source code

Name	commit
Multisig-Wallet	bc47fffb7fef907d3c5af5581c5216b8e1f3eb11

```
contracts/
└── CommonMultiSigWallet.sol

1 directory, 1 file
```

RISK CLASSIFICATION

Severity

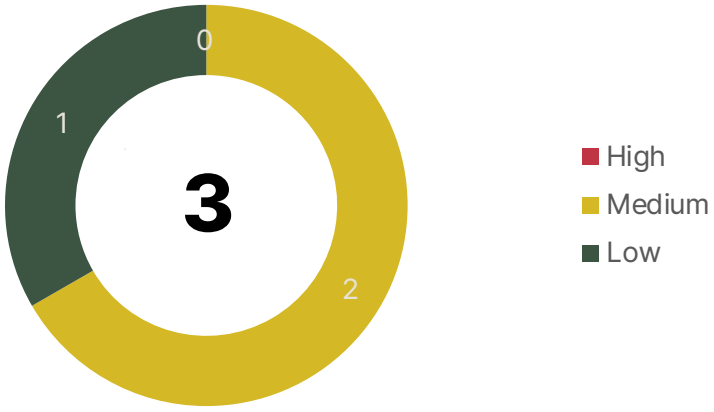
Our risk classification is based on Severity Categorization of code4ena.

> High ●
>
> Assets can be stolen, lost, compromised directly or indirectly via a valid attack path (e.g. Malicious Input Handling, Escalation of privileges, Arithmetic).

> Medium ●
>
> Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.

> Low ●
>
> Assets are not at risk. User mistake, misuse of privileges, governance risk fall under this grade.

FINDINGS BREAKDOWN



Severity	Acknowledged	fixed	Total
● High	0	0	0
● Medium	1	1	2
● Low	0	1	1
			3

* Fixed : Risk is fixed by Ozys.
* Acknowledged : Ozys has recognized the risk but has not addressed it, as it poses only a minor impact.

FINDINGS

MEDIUM

M-01. A single address can hold both owner and wallet roles

Fixed

IMPACT

`CommonMultiSigWallet` enforces a restriction preventing a single address from holding both owner and wallet roles. However, the `nextWallet` function enables bypassing this restriction.

DESCRIPTION

`addOwner` verifies that an address already registered as a `wallet` cannot be added as an owner. However, it does not check whether the address is registered as a `nextWallet`.

If an address is first registered as a `nextWallet` and then added as an `owner`, it can acquire both `wallet` and `owner` role by calling `changeWallet` on itself. The `replaceOwner` function is also affected by the same issue.

```
function addOwner(address owner)
        public
        onlyWallet
        ownerDoesNotExist(owner)
        notNull(owner)
        validRequirement(owners.length + 1, required)
    {
>>      require(owner != wallet, "Wallet cannot be owner");
        isOwner[owner] = true;
        owners.push(owner);
        emit OwnerAddition(owner);
    }
```

File 1 : CommonMultiSigWallet.sol

```
function replaceOwner(address owner, address newOwner)
        public
        onlyWallet
        ownerExists(owner)
        ownerDoesNotExist(newOwner)
    {
        require(owner != wallet, "Wallet cannot be owner");
        // 이하 생략
    }
```

File 2 : CommonMultiSigWallet.sol

When modifying `nextWallet`, `ownerDoesNotExist` check ensures that an address to be registered as a wallet is not an owner. However, the `changeWallet`, which changes the `nextWallet` to the `wallet`, only verifies that the sender is the `nextWallet` and does not check whether the address is already registered as an owner.

```
function changeNextWallet(
        address _nextWallet
>>  ) public onlyWallet ownerDoesNotExist(_nextWallet) {
        nextWallet = _nextWallet;
        emit ChangeNextWallet(_nextWallet);
    }
```

File 3 : CommonMultiSigWallet.sol

```
function changeWallet() public {
>>      require(msg.sender == nextWallet);
        emit WalletTransferred(wallet, nextWallet);

>>      wallet = nextWallet;
        nextWallet = address(0);
    }
```

File 4 : CommonMultiSigWallet.sol

The below script calls `changeNextWallet` → `addOwner` in sequence from the `wallet`, followed by calling `changeWallet` from the `nextWallet` to gain both roles.

```
const fs = require('fs');
const path = "migrations/address.json";
const ms = artifacts.require("CommonMultiSigWallet");

module.exports = function(deployer, network, accounts){
    deployer.then(async function(){
        const addr = JSON.parse(fs.readFileSync(path));
        const wallet = await ms.at(addr.Wallet.address);

        const currentWallet = accounts[3];
        const setNextWallet = await wallet.changeNextWallet(accounts[5], {from :
currentWallet});
        console.log("----------- changeNextWallet -----------",
JSON.stringify(setNextWallet.logs, null, 5));

        const newWallet = accounts[5];
        const ownerTx = await wallet.addOwner(newWallet, {from : currentWallet});
        console.log("----------- addOwner -----------", JSON.stringify(ownerTx.logs,
null, 5));

        const changeWallet = await wallet.changeWallet({from : newWallet});
        console.log("----------- changeWallet -----------",
JSON.stringify(changeWallet.logs, null, 5));
```

```
        console.log("\n\n\n[*] wallet : ", await wallet.wallet());
        console.log("[*] owners : ", await wallet.getOwners());
    })
}
```

Step to reproduce :

1. Execute truffle local node with command : `npx truffle develop`
2. Run script with command : `npx truffle migrate 1 --network develop`

The following result of executing the script is demonstraing that a single address is registered as both an `owner` and a `wallet`.

```
[*] wallet :  0xd21B0F318Ba1921F772bC6A275D95C894BfE1985
[*] owners :  [
  '0x3FDc87f4D2c3bD3c2ec67E7001ee299e8B9f7a56',
  '0xF7b2346747260ed28b3DC8Bef3ac5c0DE227D546',
  '0x4aa3643db94508c0b3FE16C0A339118D55f062C6',
  '0xd21B0F318Ba1921F772bC6A275D95C894BfE1985'
]
   ------------------------------------
   > Total cost:                  0 ETH
```

RECOMMENDATIONS

Add a check in the `require` statement of `addOwner` and `replaceOwner` to verify whether `newOwner` is a `nextWallet`, or modify `changeWallet` to ensure that `msg.sender` is not an `owner`.

STATUS Fixed

Fixed in commit `2b93f956382c24e9bf47ef9d98a33b3057ddd2e2`.

M-02. getConfirmations, getConfirmationCount function provides an incorrect confirm information

Acknowledged

IMPACT

1. When there are changes to the list of owners, `getConfirmations` returns less list of owners than it actually contains.
2. Under the same condition, `getConfirmationCount` returns a lower number of confirmations than the actual count.

DESCRIPTION

1. `getConfirmations` returns the list of owners who have confirmed the given transaction. Since it iterates over the number of owners at the time of the query, if the number of owners have decreased since the time of confirmation, it can return incorrect owner list.
   For example, suppose that there were 5 owners when the transaction with id 0 was confirmed, but only 3 owners at the time `getConfirmations` called. Even though 5 owners originally confirmed the transaction, the function may return 3 or fewer list of owners.

```
 function getConfirmations(uint256 transactionId)
        public
        view
        returns (address[] memory _confirmations)
    {
        address[] memory confirmationsTemp = new address[](owners.length);
        uint256 count = 0;
        uint256 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];
    }
```

File 5 : CommonMultiSigWallet.sol

2. `getConfirmationCount` returns how many owners have confirmed the given transaction. Like the `getConfirmations`, it counts the confirmations by iterating over the number of owners at the time of the query. If the number of owners havs decreased since the time of the confirmation, it may return a lower number of counts than the actual count.

```
function getConfirmationCount(uint256 transactionId)
        public
        view
        returns (uint256 count)
    {
>>      for (uint256 i=0; i<owners.length; i++)
            if (confirmations[transactionId][owners[i]])
                count += 1;
    }
```

File 6 : CommonMultiSigWallet.sol

RECOMMENDATIONS

Add a member to the `Transaction` struct to store the number of the owners at the time of the transaction was added, and use that value as the basis for iteration.

When a new owner is added, update the owner count for transactions up to that point to reflect the increased number of owners.

STATUS Acknowledged

Ozys: Hello!

While operating the `MultiSigWallet` contract, which serves as the base for the `CommonMultiSigWallet` contract, we encountered the following situation:

- **Owner List:** A / B / C, **Required:** 2
- **Transaction 1:** Replace Owner A → D
- **Transaction 2:** Add Owner E

In this scenario, if A confirms Transactions 1 and 2, and B then confirms Transaction 1, the `confirmCount` for Transaction 2 is reset to 0. To approve Transaction 2 again, B and C (or D) would need to reconfirm.

Regarding this issue, we believe the following:

- It is more accurate to determine the approval status of each transaction based on the valid Owner List at the time of confirmation for that specific transaction.
- When membership changes occur (via `addOwner` or `replaceOwner`), we have maintained a policy of requiring additional confirmations for other transactions based on the updated

Owner List.

Therefore, unless there is a significant issue with the current state, we plan to retain the implementation and continue with the existing operational approach.
We would appreciate your opinion on this matter.

Thank you!

78: If that is the policy, there does not appear to be any significant issues.

● LOW

L-01. getTransactionIds : uninitialized array

Fixed

IMPACT

It is not possible to distinguish whether the returned result corresponds to Transaction 0 or if there are no transactions that meet the criteria.

DESCRIPTION

The `getTransactionIds` function returns a list of transaction IDs within a specified range that meet the `pending` or `executed` criteria. Here, `pending` refers to transactions that are still awaiting confirmation, while `executed` refers to transactions that have been successfully completed.

The filtered transaction IDs are stored in the `_transactionIds` array, which is created with a size of `to - from` and then returned. However, `_transactionIds` is not explicitly initialized after its creation. In Solidity, the default value for `uint256` arrays is `0`, meaning that if no transactions meet the specified criteria, `_transactionIds` will be returned filled with zeros.

The issue with this vulnerability is that transaction IDs start from `0`, making it impossible to distinguish between a case where a transaction with `ID = 0` is returned and a case where no transactions meet the criteria.

For example, in a scenario where only one transaction exists and it meets the specified criteria, `[0]` would be returned, indicating the transaction with `ID = 0`. However, if no transactions meet the criteria, the default `[0]` array is also returned, leading to ambiguity.

```
function getTransactionIds(uint256 from, uint256 to, bool pending, bool executed)
        public view returns (uint256[] memory _transactionIds)
    {
        uint256[] memory transactionIdsTemp = new uint256[](transactionCount);
        uint256 count = 0;
        uint256 i;
        for (i=0; i<transactionCount; i++)
            if (   pending && !transactions[i].executed
                || executed && transactions[i].executed)
            {
                transactionIdsTemp[count] = i;
                count += 1;
            }
>>      _transactionIds = new uint256[](to - from);
>>      for (i=from; i<to; i++)
            _transactionIds[i - from] = transactionIdsTemp[i];
    }
```

File 7 : CommonMultiSigWallet

RECOMMENDATIONS

Return an empty array (`[]`) if no transactions meet the criteria to prevent confusion with an actual transaction ID of 0

STATUS Fixed

Ozys : Hello! The getTransactionIds logic has been revised to iterate over the range [from, to) indices. Your review would be greatly appreciated.

Thank you!

78 : The `transactionIdsTemp`, which temporarily stores the filtered results, originally had the size of the entire contract's transaction length but was modified to have a size of `to - from` after the patch. Consequently, the iteration count changed from iterating through all existing transactions to iterating `to - from` times.

In Solidity, when accessing an index in a mapping that does not exist, it does not throw an error but instead returns an element where all members are initialized to the default values of their respective types.

As a result, after the patch, when filtering by `pending`, even if no transactions exist, the `executed` member of `transactions[i]` will always have its default value of `false`, satisfying the condition. This causes `to - from` transaction IDs to be returned regardless of the existence of actual transactions. Thus, it becomes impossible to determine whether the returned transaction IDs actually exist.

```
function getTransactionIds(uint256 from, uint256 to, bool pending, bool executed)
        public view
        returns (uint256[] memory _transactionIds)
    {
-        uint256[] memory transactionIdsTemp = new uint256[](transactionCount);
+        uint256[] memory transactionIdsTemp = new uint256[](to - from);
        uint256 count = 0;
        uint256 i;
-        for (i=0; i<transactionCount; i++)
-            if (   pending && !transactions[i].executed
+        for (i=from; i<to; i++)
+            if (pending && !transactions[i].executed
                || executed && transactions[i].executed)
            {
                transactionIdsTemp[count] = i;
                count += 1;
            }
-        _transactionIds = new uint256[](to - from);
-        for (i=from; i<to; i++)
-            _transactionIds[i - from] = transactionIdsTemp[i];

+        _transactionIds = new uint256[](count);
```

```
+        for (i=0; i<count; i++)
+            _transactionIds[i] = transactionIdsTemp[i];
    }
```

File 8 : CommonMultiSigWallet

Even when there are no transactions, querying indices 100 to 103 still results in the IDs 100, 101, and 102 being returned.

```
truffle(develop)> const w = artifacts.require("CommonMultiSigWallet");
undefined
truffle(develop)> let address = JSON.parse(fs.readFileSync("migrations/address.json"));

undefined
truffle(develop)> let wallet = await w.at(address.Wallet.address);
undefined
truffle(develop)> wallet.getTransactionCount(1,0)
BN { negative: 0, words: [ 0, <1 empty item> ], length: 1, red: null }
truffle(develop)> wallet.getTransactionIds(100, 103, 1, 0)
[
  BN {
    negative: 0,
    words: [ 100, <1 empty item> ],
    length: 1,
    red: null
  },
  BN {
    negative: 0,
    words: [ 101, <1 empty item> ],
    length: 1,
    red: null
  },
  BN {
    negative: 0,
    words: [ 102, <1 empty item> ],
    length: 1,
    red: null
  }
]
truffle(develop)>
```

We recommend restricting the query range by enforcing `to <= transactionCount` and `from < to` to ensure it does not exceed the actual number of transactions.

Ozys : Fixed in commit `0cad9d5700d8783e7b317a43737e18c7711660f2`.

ABOUT 78ResearchLab

78ResearchLab is a offensive security corporation offering security auditing, penetration testing, education to enterprises, national organizations, and laboratories with the goal of making safe and convenience digital world. We have our own proprietary technology from system/security analysis and projects on various industries. We are working with the top technical experts who have won prizes in global Realword Hacking Competition/CTF, reported numerous security vulnerabilities, and have 10 years of experience in the information security.
Learn more about us at https://www.78researchlab.com/.

ABOUT RED SPIDER

Red Spider offers cyber security research and auditing service with our new R&D technologies and customized solution in IoT, OS, Web3.
Learn more about red spider at https://www.78researchlab.com/redSpider.html.