

Blockchain and Smart Contracts Using Ethereum

Contents

1	Introduction	1
1.1	Goals	1
2	Methodology	1
2.1	Protection Against Replay Attacks	1
2.2	n-of-m Multi-Signature Wallet	1
2.3	Checking Sufficiency of Contract Balance	2
2.4	Retrieving Owners of Transaction Signatures	2
3	Discussion	2
3.1	Resources Used in Deployment	2
3.2	Full Receipts of Confirmation and Submission Calls	4
4	Conclusion	5
5	Source Code	6

1 Introduction

An incompletely implemented Ethereum smart contract that implements a multi-signature wallet was provided.

1.1 Goals

- Add functionality to the smart contract to protect against replay attacks
- Change the wallet to support an arbitrary number of owners (m) with a minimum required number of signers (n)
- Include a check for sufficient contract balance before executing transfer of Ether
- Create a function that retrieves the addresses of owners that signed a transaction
- Discuss the contents of transaction receipt logs

2 Methodology

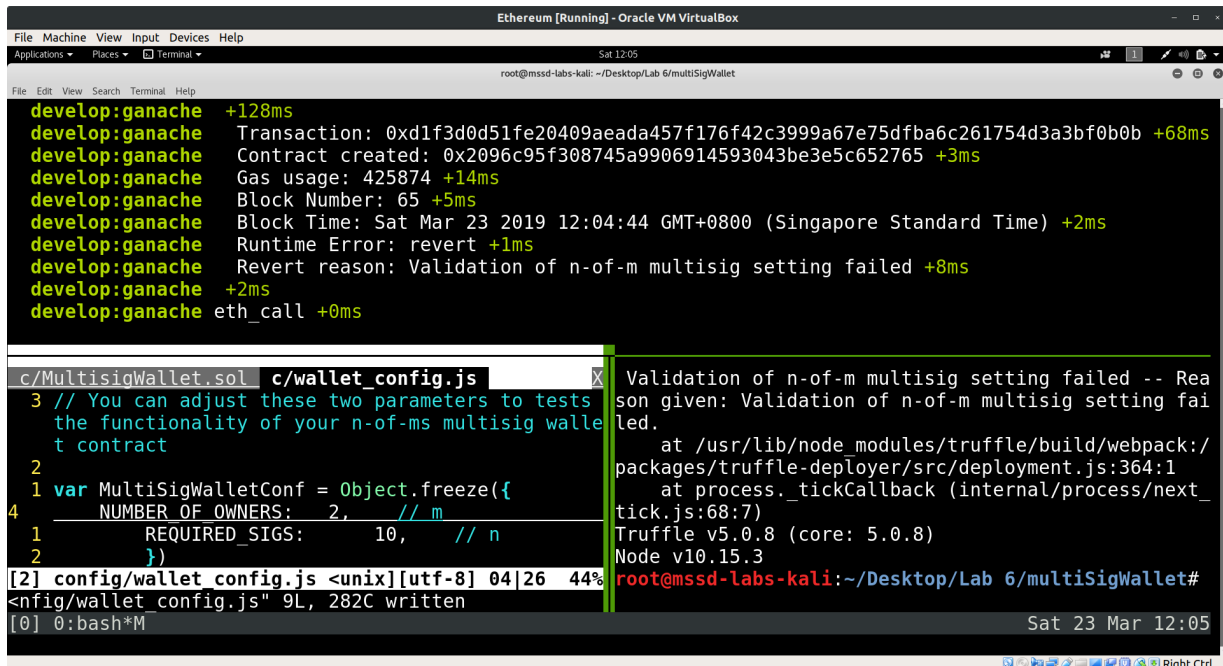
2.1 Protection Against Replay Attacks

An additional `executed` boolean field was added to `struct Transaction`. The initial value of this field is `false` when the transaction is first added, and is flipped to `true` after the transaction is executed. A `txnNotExecuted` modifier is created and placed at the start of the `executeTransaction()` function. The modifier will `revert` if the transaction has already been executed, preventing the replay from occurring.

2.2 n-of-m Multi-Signature Wallet

The wallet was changed from a 2-of-2 scheme to an n-of-m scheme. Loops were used to check that each owner of the contract was not a null address, and that no repeated owners were included. Once the checks pass, another loop was used to save the owners of the contract.

The modifier `checkValidSettings` was also changed to test for valid setting ranges. `ownerCount` should not be zero (0) or exceed `MAX_OWNER_COUNT`. `_requiredSigs` should not be zero (0) or exceed `ownerCount` (instead of `MAX_OWNER_COUNT` as stated in the task requirement, in order to catch invalid settings where the required number of signatures exceeds the number of owners of the contract.)



```

Ethereum [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places Terminal
Sat 12:05
root@mssd-labs-kali: ~/Desktop/Lab 6/multiSigWallet

develop:ganache +128ms
develop:ganache Transaction: 0xd1f3d0d51fe20409aeada457f176f42c3999a67e75dfba6c261754d3a3bf0b0b +68ms
develop:ganache Contract created: 0x2096c95f308745a9906914593043be3e5c652765 +3ms
develop:ganache Gas usage: 425874 +14ms
develop:ganache Block Number: 65 +5ms
develop:ganache Block Time: Sat Mar 23 2019 12:04:44 GMT+0800 (Singapore Standard Time) +2ms
develop:ganache Runtime Error: revert +1ms
develop:ganache Revert reason: Validation of n-of-m multisig setting failed +8ms
develop:ganache +2ms
develop:ganache eth_call +0ms

c/MultiSigWallet.sol c/wallet_config.js
3 // You can adjust these two parameters to tests the functionality of your n-of-m multisig wallet contract
2
1 var MultiSigWalletConf = Object.freeze({
4   NUMBER_OF_OWNERS: 2, // m
1   REQUIRED_SIGS: 10, // n
2 })
[2] config/wallet_config.js <unix>[utf-8] 04|26 44% root@mssd-labs-kali:~/Desktop/Lab 6/multiSigWallet#
<nfig/wallet_config.js" 9L, 282C written
[0] 0:bash*M
Sat 23 Mar 12:05

```

Modified task requirement to catch `_requiredSigs > ownerCount` instead of `_requiredSigs > MAX_OWNER_COUNT`.

2.3 Checking Sufficiency of Contract Balance

Additional checks for sufficient contract balance was added on transaction confirmation. A new event `NotEnoughBalance` was declared. If the requested balance (`txn.value`) exceeds the current contract balance (`address(this).balance`), the event was emitted, interrupting the execution of the transaction.

2.4 Retrieving Owners of Transaction Signatures

function `getOwnersWhoSignedTx()` was created to retrieve the addresses of the owners of transaction signatures. Initially, `push()` was attempted on a new `address[]` memory array to store the addresses. This triggered a `TypeError` as the `address[]` array is of fixed length when declared in memory instead of storage. Instead, the owner addresses had to be assigned individually to indexes of the array of length `getSignatureCount()`.

3 Discussion

3.1 Resources Used in Deployment

```
Ethereum [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places Terminal
root@mssd-labs-kali: ~/Desktop/Lab 6/multiSigWallet

develop:ganache eth_blockNumber +1ms [270/1692]
develop:ganache net_version +4ms
develop:ganache eth_sendTransaction +7ms
develop:ganache +139ms
develop:ganache Transaction: 0x006a7bd6a76e5368a6653760ce110885e05338efc90131a54587a8563344cfe7 +2ms
develop:ganache Contract created: 0x463794408df63f92d203a3850c92279d440baa48 +1ms
develop:ganache Gas usage: 1847465 +1ms
develop:ganache Block Number: 3 +0ms
develop:ganache Block Time: Sat Mar 23 2019 12:22:52 GMT+0800 (Singapore Standard Time) +1ms
develop:ganache +0ms
develop:ganache eth_getTransactionReceipt +1ms

c/MultiSigWallet.sol c/wallet_config.js
3 // You can adjust these two parameters to tests
  the functionality of your n-of-ms multisig wallet
  contract
2
1 var MultiSigWalletConf = Object.freeze({
4   NUMBER_OF_OWNERS: 10, // m
1   REQUIRED_SIGS: 10, // n
2 })
[2] config/wallet_config.js <unix>[utf-8] 04|26 44%
<nfig/wallet config.js" 9L, 283C written
[0] 0:[tmux]*M

executed transaction again. (93ms)
✓ Check event NotEnoughBalance is emitted properly. (897ms)
✓ Get owners who signed the 2nd transaction (i.e. ID = 1). (88ms)
8 passing (4s)

root@mssd-labs-kali:~/Desktop/Lab 6/multiSigWallet#
Sat 23 Mar 12:25
```

The contract was created in block number 3 (assuming initial run of `truffle develop --log`).

The gas cost for contract deployment varies depending on the number of owners and required signatures. For a multi-signature smart contract with two owners, the gas cost is 1,498,037. For a smart contract with 10 owners, the gas cost rises to 1,847,465.

At the time of this report (2019 Mar 23 0510 UT), assuming a gas price of 2 Gwei (2e-9 ether) for a standard wait time of less than 5 minutes, 1,847,465 gas converted to USD\$0.506.¹

ETH Gas Station

GENERAL

Main Page

Tx Calculator

TxPool Vision

Low Gas Price Watch List

Gas Burners

FAQ

External Links

Estimates over last 1,500 blocks - Last update: Block 7423021

Change Currency

Std Cost for Trans...

\$0.006

Gas Price Std (G...

2

SafeLow Cost for ...

\$0.003

Gas Price SafeLo...

1

Median Wait (s)

27

Median Wait (bloc...

2

Gas-Time-Price Estimator: For transactions sent at block: 7423021

Adjust confirmation time

Avg Time (min)

2.59

95% Time (min)

6.48

Gas Price (Gwei)*

2

Tx Fee (Fiat)

\$0.506

Gas Used*

1847465

Avg Time (blocks)

12.2769871764

95% Time (blocks)

30.692467941

Tx Fee (ETH)

0.00369

Real Time Gas Use: % Block Limit (last 10)

Last Block: 7423021

Transaction Count by Gas Price

Confirmation Time by Gas Price

Speed

Gas Price (gwei)

SafeLow (<30m)	1
Standard (<5m)	2
Fast (<2m)	4

Note: Estimates not valid when multiple transactions are batched from the same address or for transactions sent to addresses with many (e.g. > 100) pending

Top 10 Miners by Blocks Mined: Support for user transactions

Miner	Lowest gas price (gwei)	Weighted avg gas price (gwei)	% of total blocks
0xd224ca0c819e8e97ba0136b3b95ceff503b79f53	0	8	2
miningpoolhub	0.12	13	4
Nanopool	0.4	8	11
Dwarfpool	1	9	2
0x2a5994b501e6a560e727b6c2de5d856396aadd38	1	9	2
Ethermine	1	10	28
0x5a0b54d5dc17e0aad3383d2db43b0a0d3e029c4c	1	11	27
0x09ab1303d3ccaf5f018cd511146b07a240c70294	2	12	2
0x005e288d713a5fb3d7c9cf1b43810a98688c7223	2	13	1
f2pool	3	15	11

Misc Stats (Last 1,500 blocks)

Category	Value
Cheapest Gas Price (gwei)	0
Highest Gas Price (gwei)	134
Median Gas Price (gwei)	5
Cheapest Transfer Fee	\$0.0029
Highest Transfer Fee	\$0.29
Total Transactions	1160
% Empty Blocks	1
% Full Blocks	46

ETH Tips - Thank you!

0x446fa0c8EaD753c7ABf0B821f90D4338e72De380

Try the safelow gas price with metamask or mist

Gentelella - Bootstrap Admin Template by Colorlib

3.2 Full Receipts of Confirmation and Submission Calls

Since the output of `contract.submitTransaction` was stored in `var receipt`, `console.log` was called on `receipt`:

```
var receipt = await contract.submitTransaction(RECEPIENT, VALUE_SENT, {from: accounts[0]});
console.log(receipt);
```

This resulted in the output detailed within [receipt.log](#).

The extracted log contains details on the *Submission* event, as well as the *Confirmation* event.

The fields on the full receipt log as follows:

tx:	Contains the hash string for the transaction.
receipt.transactionHash:	Contains the hash string for the transaction.
receipt.transactionIndex:	Indicates the position of the transaction in the block.
receipt.blockHash:	Hash string of the block containing the transaction.
receipt.blockNumber:	Block number containing the transaction.
receipt.from:	Address of the sender.
receipt.to:	Address of the receiver.
receipt.gasUsed:	Amount of gas used by this transaction.
receipt.cumulativeGasUsed:	Total amount of gas used when this transaction was executed in the block.
receipt.contractAddress:	Address of contract if the transaction is a contract creation.
receipt.logs:	Array of log objects generated by this transaction.
receipt.status:	Boolean to indicate transaction failure or success.
receipt.logsBloom:	Bloom filter for the logs of this block.
receipt.v:	Recovery ID (recid) for public key recovery. Used to speed up signature verification.
receipt.r:	Part of the Elliptic Curve Digital Signature Algorithm (ECDSA) signature pair, used in public key recovery.
receipt.s:	The other part of the ECDSA signature pair, used in public key recovery.
receipt.rawLogs:	Undecoded logs; stopgap measure until Application Binary Interface (ABI) for all events can be obtained
logs.logIndex:	Log position in the block.
logs.transactionIndex:	Indicates the position of the transaction in the block that this log was created from.
logs.transactionHash:	Contains the hash string for the transaction that this log was created from.
logs.blockHash:	Hash string of the block containing this log.
logs.blockNumber:	Block number containing this log.
logs.address:	Address where this log originated.
logs.type:	<i>'pending'</i> or <i>'mined'</i> .
logs.id:	Log identifier.
logs.event:	The name of the event that triggered this log.
logs.args:	The arguments from this event.

4 Conclusion

The 2-of-2 multi-signature smart contract wallet was converted to support n-of-m signatures. The functionality for protection against replay attacks was added, as well as a check for sufficient balance before transfer. A new function was added that retrieves addresses of owners who signed a transaction. The full transaction receipt logs were examined.

5 Source Code

MultisigWallet.sol

1

<https://ethgasstation.info/>