BUG REPORT : MultisigWallet vulnerable to unconfirmed transactions

Author : Philippe Castonguay

Date : 15.04.2018

Short Description : Multisignature wallet is vulnerable to involuntary pending transaction confirmation status. Certain actions can lead to involuntary change of pending transaction status, transactions that could at a given time point (e.g. far future) be considered malicious or undesired.

Description

The Golem project is using an early implementation of Gnosis' MultiSigWallet.sol contract, which can be found here ; https://etherscan.io/address/0x7da82c7ab4771ff031b66538d2fb9b0b047f6cf9#code.

The transaction confirmation workflow is as follow ;

1. `owner` A calls `submitTransaction()` to add a new transaction `TX_A`. `TX_A` has 1 confirmation.
2. Another `owner` calls `confirmTransaction(TX_A)`. `TX_A` has 2 confirmation.
3. Step **2** is reproduced by different owners until `isConfirmed(TX_A)` returns **true**.
4. **Anyone** can call `executeTransaction(TX_A)`, which executes `TX_A`.

The `isConfirmed(TX_HASH)` function returns true when the number of confirmations for `TX_HASH` is higher than `required`, a variable set by the owners of the multisig.

The vulnerability in the current implementation is two fold. First of all, **an unconfirmed transaction can become confirmed if `required` is changed**. This could happen if `updateRequired(_newRequiredValueSmallerThanPrevious)` is called or if `removeOwner(_ownerToRemove)` is called and the new list of owner would be smaller than `required`. The second part of this vulnerability is that **anyone can execute a confirmed transaction**. Indeed, in Golem's MultiSignature contract version, `executeTransaction(TX_HASH)` is not restricted to the contract owners, allowing anyone to execute a confirmed & un-executed transaction. This latter point was fixed in a later Gnosis Multisignature contract version (see https://github.com/gnosis/MultiSigWallet/blob/a227d6e342fa438b9e956d6c32b70c86678eabbc/contracts/MultiSigWallet.sol#L224). This means that if there is a malicious transaction that is still pending (e.g. owner A tried to add address D to the owners list 5 years ago) and if this transaction is unintentionally confirmed by calling `updateRequired()` or `removeOwner()`, the transaction has a risk of being executed, **even within the same block**.

There are currently **7 pending, unconfirmed transactions** on the Golem Multisignature contract. **3/7** unconfirmed transaction **have 2 confirmations**, just 1 below the current `required` number of confirmation required (3). None of these transactions look dangerous, but they could be if one of the owner is malicious or *becomes* malicious. Indeed, a now honest owner could send a transaction requesting to add his ledger wallet as an owner and remove his current address from the owner list. If this transaction is forgotten (or is hidden, see below) and if this owner is removed after being in conflict with the team, this transaction could be executed in the future involuntary, giving back ownership to the now expelled owner.

Hidden `submitTransaction()` Calls

Golem team can not rely on the transactions seen on Etherscan to notice new transactions being submitted. Indeed, it is possible to call `submitTransaction()` without leaving any trace on Etherscan. This is caused by Etherscan not tracking all internal transactions. For instance, the following contract calls the Golem Multisig without leaving any traces on Etherscan ;

```
contract InternalPass {

    MultiSigWallet multiSig;

    //Golem Multisig contract instance
    function InternalPass(){
        multiSig = MultiSigWallet(0x7da82C7AB4771ff031b66538D2fB9b0B047f6CF9);
    }

    //Submit a transaction to Golem Multisig via internal call
    function sendInternal() public {
         multiSig.submitTransaction(address(multiSig), 0, '0xThisIsPhilippe', 0x666);
    }

}
```

This call would fail if the contract is not a Golem Multisig `owner`, which was the case when I tested it, but the failed `submitTransaction()` left no traces on the Golem Multisig etherscan page. One can compare the blocknumber and timestamp of the `sendInternal()` transaction with the transactions shown on the Golem Multisig address and see that the `submitTransaction()` is indeed not shown ;

Hidden `submitTransaction()` call transaction ;
https://etherscan.io/tx/0xf8a545a48353a5785a6ad891a00f5e99abcecc00e7a2c6ff96e85fdd3add665d

Golem Multisig internal transactions ;
https://etherscan.io/address/0x7da82C7AB4771ff031b66538D2fB9b0B047f6CF9#internaltx

One could also call `executeTransaction()` (or any other function) in a similar manner and the the transaction would be executed silently.

Attack

If the `required` variable is changed intentionally (`updateRequired()`) or not (`removeOwner()`), it is possible that a forgotten and now malicious transaction be executed. This transaction could've been honest at one point in time, but could eventually, if executed much later, comprise the entire integrity of the multisig wallet.

Impact: *High*

Difficulty : *High*

Components:

- **MultiSigWallet.sol**

Attack Scenarios

Honest ownership transfer

Bob is one of the owner of the Multisig contract in 2018 and is honest. Bob wants to transfer his ownership to his new Ledger address and calls `submitTransaction()` for this. Bob talks about it in a meeting, others confirm it, but Bob doesn't execute it as he forgets. Bob is fired later in 2018 and his ownership of the multisig is revoked. Bob knows he has a pending transaction and wait for it to be validated by accident (see #description section). Bob sets a script that secretly executes his transaction whenever the transaction becomes valid. In 2020, Bob's ownership transfer transaction becomes valid, Bob's script execute secretly the transaction and Bob is now an owner of the contract again, without the other owners being able to tell.

A similar scenario happened to the 0x Team :

I recently notified the 0x team that their first submitted transaction could be executed by anyone, but hasn't yet. The `submitTransaction()` was called **245 days** before it was executed, as it was forgotten. They executed the transaction 6 days ago after I noticed them (they did it with an internal call).

Here's the discussion ;

> PhABC :
>
> Hey Amir, was going through the proxy contract and noticed the multisig had a pending tx (tx 0) that is ready to be executed for quite some time now, replacing owner `0x58f59df649fcff096f3fa49d289aa7f5a2a0fb33` by `0x257619b7155d247e43c8b6d90c8c17278ae481f0`. Not sure if problematic or voluntary, but since anyone can call the `executeTransaction()` function, thought it wouldn't hurt to confirm with you.
>
> Amir :
>
> 0x Team 3:03 PM
>
> hey @PhABC , thanks for pointing this out. We must have created a transaction to replace an address and forgotten to execute it after the time lock - but I can confirm we own both addresses

Honest owner fund transfer

A similar scenario could happen where the owners initially want to move their fund to a new address but decide otherwise, without canceling the pending transactions. A few years later, the transaction becomes valid unintentionally. It could maliciously be executed to either lock the funds forever or transfer the funds to a past member of the Golem team.

Reproduction:

No reproduction script since is human error based, an error greatly facilitated by the contract design.

Fixes

1. Deploy a new Multisig and transfer the funds

The most secure option would be to deploy a new multisignature wallet that is updated and contains additional security gates to prevent the attacks presented in this document. These additional security gates could be as follow ;

One solution would be to allow multisig owners to set a time limit after which transactions aren't valid, i.e. an expiration time for pending transaction. Note the below scheme refers to the most recent Gnosis MultisigWallet.sol version ;

```
// Reorded for storage efficiency and added 'expirationTime'
struct Transaction {
    bool executed;
    address destination;
    uint expirationTime;
    uint value;
    uint nonce;
    bytes data;
}

...

modifier notExpired(bytes32 transactionHash) {
        require(now < transactions[transactionHash].expirationTime);
        _;
}

...

function executeTransaction(bytes32 transactionHash)
        public
        ownerExists(msg.sender)
        confirmed(transactionId, msg.sender)
        notExecuted(transactionHash);
        notExpired(transactionHash);            ///// <------ Added modifier
{
        ...
}
```

`expirationTime` can be set by the smart contract when adding a new transaction, based on a variable like `timeToExpire`, where `expirationTime: now.add(timeToExpire)` when calling `addTransaction(...)`.

Another solution would be to **not** allow any function that update the `required` variable if there are pending transactions. This could be implemented with ;

```
function updateRequired(uint _required)
    public
    onlyWallet
    validRequirement(owners.length, _required)
{
    // Number of pending transactions
    bytes32[] pendingTransactions = getPendingTransactions();

    //Throw if any pending transactions
    require(pendingTransactions.length == 0);
    required = _required;
    RequirementChange(_required);
}
```

The only problem I have with this approach is that a malicious owner could prevent the other owners from removing them by calling `submitTransaction()` right before the others try to remove them as a owner. Of course, this is not problematic in practice as the other owners just need to be successful once, but it could lead to some annoyances.

2. Delete every pending transaction periodically

The previous suggestion is more secure and permanent, but can be seen as more complex. Another solution would be to delete all currently pending transactions and periodically repeat this process. This solution requires more long term effort and will depend on the continual diligence of the developing team. Luckily, the `getPendingTransactions()` can easily show whether any transaction is pending or not, even if this transaction was added via a "hidden" `submitTransaction()` call.