Endorsement Policy working in Hyperledger Fabric v1.0.4

Objective:

To verify whether the endorsement policy evaluation is done by Fabric (VSCC) properly or not at the time of commit, when **insufficient endorsements** are presented from client.

This tests were performed on fabric and fabric-ca v1.0.4 version, and fabric-samples/balance-transfer release version (b99e1b8da2bc4d17f689c152078d96e8f6428a27). Applied patch as in `endorse_test_code_changes_v1.0.4.txt` attachment to pass endorsement request to endorsement peers from all organizations (which needs to endorse the transaction) from the client side.

Need for patch: From balance-transfer code exploration, it is found that existing helper.js code (for loop, where clients, channels are initialized) maintains separate client objects for each organization, in which channel object is present and to the channel object organization peers are added. Due to which target peers are populated only for that organization. Events from second organization peers are not considered here, to avoid MSP related error.
p.s: The patch is only a tweak to test endorsements and may not be a proper (implementation as in v1.1.0-preview).

Applying patch: To apply `endorse_test_code_changes_v1.0.4.txt`, below commands can be used. Similar changes can be applied to test endorsement policy for fabric v1.0.0 to v1.0.4.

```
cd fabric-samples/balance-transfer
patch -p2 < endorse_test_code_changes_v1.0.4.txt
```

Tests:

1. Policy where endorsement from peers of two different organizations are required. (**AND condition**)

The below epolicy need to be specified as part of instantiate request. Patch also has the relevant code.

```
var epolicy = {
    identities: [
        { role: { name: "member", mspId: "Org1MSP" }},
        { role: { name: "member", mspId: "Org2MSP" }}
    ],
    policy: {
        "2-of": [{ "signed-by": 0 }, { "signed-by": 1 }]
    }
};

var request = {
    chaincodeId: chaincodeName,
    chaincodeVersion: chaincodeVersion,
    args: args,
    txId: tx_id,
    "endorsement-policy": epolicy
};
```

Test Case 1. Sending transaction proposal request to only one endorsing peer from one of the organization peers.
Simulation:

```
curl -s -X POST http://localhost:4000/channels/mychannel/chaincodes/mycc -H "authorization: Bearer
$ORG1_TOKEN" -H "content-type: application/json" -d '{ "fcn":"move", "args":["a","b","10"], "peers":
["peer1"]}'
```

Here, `ORG1_TOKEN` is the login token returned on user (eg., Jim) enrollment to organization1. Here, Token decides the organization name as per the balance-transfer code.

Note: In the above command, we have passed only peers argument and ignored other_args argument, to simulate sending transaction proposal request to only single organization peer.

Results:
-- this test failed as expected
-- the VSCC thrown error
-- the balance was unaffected by this transaction

```
curl -s -X GET "http://localhost:4000/channels/mychannel/chaincodes/mycc?peer=peer1&fcn=query&args=%5B
%22a%22%5D" -H "authorization: Bearer $ORG1_TOKEN" -H "content-type: application/json"
```

Test Case 2. Sending transaction proposal request to two endorsing peers from the same organization.
Simulation:

```
curl -s -X POST http://localhost:4000/channels/mychannel/chaincodes/mycc -H "authorization: Bearer
$ORG1_TOKEN" -H "content-type: application/json" -d '{ "fcn":"move", "args":["a","b","10"], "peers":
["peer1",”peer2”]}'
```

Note: In the above command, we have passed 2 peer names and ignored other_args argument, to simulate sending transaction proposal request to 2 peers of the same organization.

Results:
-- this test failed as expected
-- the VSCC thrown error
-- the balance was unaffected by this transaction

Test Case 3. Sending transaction proposal request to two endorsing peers from different organizations.
Simulation:

```
curl -s -X POST http://localhost:4000/channels/mychannel/chaincodes/mycc -H "authorization: Bearer
$ORG1_TOKEN" -H "content-type: application/json" -d '{ "fcn":"move", "args":["a","b","10"], "peers":
["peer1"], "other_orgs":["org2"]}'
```

Note: In the above command, we have passed 1 peer name in the peers argument and 'org2' as other_args argument, to simulate sending transaction proposal request to second organization peer as well. i.e. 1 peer from org1 and org2 are sent request.

Results:
-- this test succeeded as expected
-- the balance was updated after this transaction

2. Policy where endorsement from one of the two organizations is sufficient. (**OR condition**). This is the default policy implemented (even if we do not specify epolicy as below this endorsement policy applies)

```
var epolicy = {
    identities: [
        { role: { name: "member", mspId: "Org1MSP" }},
        { role: { name: "member", mspId: "Org2MSP" }}
    ],
    policy: {
        "1-of": [{ "signed-by": 0 }, { "signed-by": 1 }]
    }
};

var request = {
    chaincodeId: chaincodeName,
    chaincodeVersion: chaincodeVersion,
    args: args,
    txId: tx_id,
    "endorsement-policy": epolicy
};
```

Test Case 1. Send transaction proposal request to only one peer from one of the organizations
Simulation: Same as in Test Case 1 of AND policy testing

Results:
-- this test succeeded as expected
-- the balance was updated after this transaction

Conclusion: Test results show that the endorsement policy verification done by the peers as expected.

Objective 2:

To verify the use of **status information** present in endorsing peer's proposal response object.

Test Case 1. Change status from 200 (ok) to 500 (not ok) and By-pass client side (SDK and client app) checks, so that request is sent to Fabric

Setup: Code changes to By-pass client side checks
1. Edit `sendTransaction()` code in `node_modules/fabric-client/lib/Channel.js` as below. i.e. avoid checks on status - comment lines 1451, 1454, 1457, 1459

```
var endorsements = [];
let proposalResponse = proposalResponses;
if(Array.isArray(proposalResponses)) {
    for(let i=0; i<proposalResponses.length; i++) {
        // make sure only take the valid responses to set on the consolidated response object
        // to use in the transaction object
        //if (proposalResponses[i].response && proposalResponses[i].response.status === 200) {
            proposalResponse = proposalResponses[i];
            endorsements.push(proposalResponse.endorsement);
        //}
    }
} else {
    //if (proposalResponse && proposalResponse.response && proposalResponse.response.status === 200) {
        endorsements.push(proposalResponse.endorsement);
    //}
}
```

2. Change response status in app/invoke-transaction.js , just after proposalResponses inspection, which checks for all good

```
proposalResponses[0].response.status = 500;
```

Simulation:
Issue below Invoke command

```
curl -s -X POST http://localhost:4000/channels/mychannel/chaincodes/mycc -H "authorization: Bearer
$ORG1_TOKEN" -H "content-type: application/json" -d '{ "fcn":"move", "args":["a","b","10"], "peers":
["peer1"], "other_orgs":["org2"]}'
```

Results:
-- this test succeeded as expected
-- the transaction goes through fine in the backend and gets committed

Conclusion: The **status information** in the transaction proposal response from endorsers, is checked at the client side only (SDK, client script) and it is not verified by the committers before committing. If we modify the status in the responses from 200 (ok) to 500 (not ok), and by-pass the client side checks, the transaction goes through fine in the backend and gets committed.