```solidity
pragma solidity ^0.4.0;

contract AccessControlMethod{

    address public owner;
    address public subject;
    address public object;
    Judge public jc;

    event ReturnAccessResult(
        address indexed _from,
        string  _errmsg,
        bool _result,
        uint _time,
        uint _penalty
    );

    struct Misbehavior{
        string res;                //resource on which the misbehavior is conducted
        string action;             //action (e.g., "read", "write", "execute") of the misbehavior
        string misbehavior;        //misbehavior
        uint time;                 //time of the misbehavior occured
        uint penalty;              //penalty opposed to the subject (number of minutes blocked)
    }

    struct BehaviorItem{           //for one resource
        Misbehavior [] mbs;        //misbehavior list of the subject on a particular resource
        uint TimeofUnblock;        //time when the resource is unblocked (0 if unblocked; otherwise, blocked)
    }

    struct PolicyItem{            //for one (resource, action) pair;
        bool isValued;            //for duplicate check
        string permission;        //permission: "allow" or "deny"
        uint minInterval;         //minimum allowable interval (in seconds) between two successive requests
        uint ToLR;                //Time of Last Request
        uint NoFR;                //Number of frequent  Requests in a short period of time
        uint threshold;           //threshold on NoFR, above which a misbehavior is suspected
        bool result;              //last access result
        uint8 err;                //last err code
    }

    mapping (bytes32 => mapping(bytes32 => PolicyItem)) policies;  //mapping (resource, action) => PolicyCriteria for policy check
    mapping (bytes32 => BehaviorItem) behaviors;   //mapping resource => BehaviorCriteria for behavior check

    /*convert strings to byte32*/
    function stringToBytes32(string _str) public constant returns (bytes32){
        bytes memory tempBytes = bytes(_str);
        bytes32 convertedBytes;
        if(0 == tempBytes.length){
            return 0x0;
        }
        assembly {
            convertedBytes := mload(add(_str, 32))
        }
        return convertedBytes;
    }

    function AccessControlMethod(address _subject) public{
        owner = msg.sender;
        object = msg.sender;
        subject = _subject;
    }

    function setJC(address _jc)public{
        if(owner == msg.sender){
            jc = Judge(_jc);
        }
        else throw;

    }

    function policyAdd(string _resource, string _action, string _permission, uint _minInterval, uint _threshold) public{
        bytes32 resource = stringToBytes32(_resource);
        bytes32 action = stringToBytes32(_action);
        if(msg.sender == owner){
            if(policies[resource][action].isValued) throw; //duplicated key
            else{
                policies[resource][action].permission = _permission;
                policies[resource][action].minInterval = _minInterval;
                policies[resource][action].threshold = _threshold;
                policies[resource][action].ToLR = 0;
                policies[resource][action].NoFR = 0;
                policies[resource][action].isValued = true;
                policies[resource][action].result = false;
                behaviors[resource].TimeofUnblock = 0;
            }
        }
        else throw;
    }

    function getPolicy(string _resource, string _action) public constant returns (string _permission, uint _minInterval, uint _threshold, uint _ToLR, uint _NoFR, bool _res, uint8 _errcode){
        bytes32 resource = stringToBytes32(_resource);
        bytes32 action = stringToBytes32(_action);
        if(policies[resource][action].isValued){
            _permission = policies[resource][action].permission;
            _minInterval = policies[resource][action].minInterval;
            _threshold = policies[resource][action].threshold;
            _NoFR = policies[resource][action].NoFR;
            _ToLR = policies[resource][action].ToLR;
            _res = policies[resource][action].result;
            _errcode = policies[resource][action].err;
        }
        else throw;

    }

    function policyUpdate(string _resource, string _action, string _newPermission) public{
        bytes32 resource = stringToBytes32(_resource);
        bytes32 action = stringToBytes32(_action);
        if(policies[resource][action].isValued){
            policies[resource][action].permission = _newPermission;
        }
        else throw;
    }

    function minIntervalUpdate(string _resource, string _action, uint _newMinInterval) public{
        bytes32 resource = stringToBytes32(_resource);
        bytes32 action = stringToBytes32(_action);
        if(policies[resource][action].isValued){
            policies[resource][action].minInterval= _newMinInterval;
        }
        else throw;
    }

    function thresholdUpdate(string _resource, string _action, uint _newThreshold) public{
        bytes32 resource = stringToBytes32(_resource);
        bytes32 action = stringToBytes32(_action);
        if(policies[resource][action].isValued){
            policies[resource][action].threshold= _newThreshold;
        }
        else throw;
    }

    function policyDelete(string _resource, string _action) public{
        bytes32 resource = stringToBytes32(_resource);
        bytes32 action = stringToBytes32(_action);
        if(msg.sender == owner){
            if(policies[resource][action].isValued){
                delete policies[resource][action];
            }
            else throw;
        }
        else throw;
    }

    /*Use event*/
    function accessControl(string _resource, string _action, uint _time) public{

        bool policycheck = false;
        bool behaviorcheck = true;
        uint8 errcode = 0;
        uint penalty = 0;

        if (msg.sender == subject){
            bytes32 resource = stringToBytes32(_resource);
            bytes32 action = stringToBytes32(_action);

            if(behaviors[resource].TimeofUnblock >= _time){//still blocked state
                errcode = 1; //"Requests are blocked!"
            }

            else{//unblocked state
                if(behaviors[resource].TimeofUnblock > 0){
                    behaviors[resource].TimeofUnblock = 0;
                    policies[resource][action].NoFR = 0;
                    policies[resource][action].ToLR = 0;
                }
                //policy check
                if (keccak256("allow") == keccak256(policies[resource][action].permission)){
                    policycheck = true;
                }
                else{
                    policycheck = false;
                }
                //behavior check
                if (_time - policies[resource][action].ToLR <= policies[resource][action].minInterval){
                    policies[resource][action].NoFR++;
                    if(policies[resource][action].NoFR >= policies[resource][action].threshold){
                        penalty = jc.misbehaviorJudge(subject, object, _resource, _action, "Too frequent access!", _time);
                        behaviorcheck = false;
                        behaviors[resource].TimeofUnblock = _time + penalty * 1 minutes;
                        behaviors[resource].mbs.push(Misbehavior(_resource, _action, "Too frequent access!", _time, penalty));//problem occurs when using array
                    }
                }
                else{
                    policies[resource][action].NoFR = 0;
                }
                if(!policycheck && behaviorcheck) errcode = 2; //"Static Check failed!"
                if(policycheck && !behaviorcheck) errcode = 3;  //"Misbehavior detected!"
                if(!policycheck && !behaviorcheck) errcode = 4; //"Static check failed! & Misbehavior detected!";
            }
            policies[resource][action].ToLR = _time;
        }
        else {
            errcode = 5; //"Wrong object or subject detected!";
        }
        policies[resource][action].result =  policycheck && behaviorcheck;
        policies[resource][action].err = errcode;
        if(0 == errcode) ReturnAccessResult(msg.sender, "Access authorized!", true, _time, penalty);
        if(1 == errcode) ReturnAccessResult(msg.sender, "Requests are blocked!", false, _time, penalty);
        if(2 == errcode) ReturnAccessResult(msg.sender, "Static Check failed!", false, _time, penalty);
        if(3 == errcode) ReturnAccessResult(msg.sender, "Misbehavior detected!", false, _time, penalty);
        if(4 == errcode) ReturnAccessResult(msg.sender, "Static check failed! & Misbehavior detected!", false, _time, penalty);
        if(5 == errcode) ReturnAccessResult(msg.sender, "Wrong object or subject specified!", false, _time, penalty);
    }


    function getTimeofUnblock(string _resource) public constant returns(uint _penalty, uint _timeOfUnblock){
        bytes32 resource= stringToBytes32(_resource);
        _timeOfUnblock = behaviors[resource].TimeofUnblock;
        uint l = behaviors[resource].mbs.length;
        _penalty = behaviors[resource].mbs[l - 1].penalty;
    }


    function deleteACC() public{
        if(msg.sender == owner){
            selfdestruct(this);
        }
    }
}


contract Judge{
        function misbehaviorJudge(address _subject, address _object, string _res, string _action, string _misbehavior, uint _time) public returns (uint );
}

```