Audit of the BFTG project

By OCamlPro

August 20, 2021

Table of Major and Critical Issues

Contents

Chapter 1

Overview

The Location section should be read as: The source code is available at `https://github.com/RSquad/dens-smv` at branch master with hash code equal to fbdfe4bca3c372b02cacf9788b4ad37112d0da2c and `https://github.com/RSquad/BFTG` (SMV part only) at branch master with hash code equal to 7c6ec7d811bcc1f228a3499ab19f6d20652ca94b

Chapter 2

Contract Padawan

Contents

2.1 Overview

In file `Padawan.sol`

2.2 Contract Inheritance

Base	
IEstimateVotesCallback	

2.3 Static Variable Definitions

- OK

```
32    address static _deployer;
33    address static _owner;
```

2.4 Variable Definitions

- OK

```
35    mapping(address => Balance) public _balances;
36    mapping(address => address) public _tokenAccounts;
37    mapping(address => ActiveProposal) public _activeProposals;
38    uint32 _activeProposalsLength;
40    Reclaim public _reclaim;
```

2.5 Modifier Definitions

2.5.1 Modifier onlyOwner

- OK

```
44    modifier onlyOwner() {
45        require(msg.sender == _owner, Errors.
              NOT_AUTHORIZED_CONTRACT);
46        _;
47    }
```

2.6 Constructor Definitions

2.6.1 Constructor

- OK

```
49      constructor() public onlyContract {
50          require(_deployer == msg.sender, Errors.ONLY_DEPLOYER);
51      }
```

2.7 Public Method Definitions

2.7.1 Function confirmVote

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
89      function confirmVote(
90          uint128 votes,
91          uint128 votePrice,
92          address voteProvider)
93      external onlyContract { votes;
94          optional(ActiveProposal) optActiveProposal =
                _activeProposals.fetch(msg.sender);
95          require(optActiveProposal.hasValue(), 111);
96          uint128 activeProposalVotes = optActiveProposal.get().votes
                ;
97
98          address balanceProvider = voteProvider == address(0) ?
                voteProvider : _tokenAccounts[voteProvider];
99
100         if(_balances[balanceProvider].locked < (activeProposalVotes
                ) * votePrice) {
101             _balances[balanceProvider].locked = (
                    activeProposalVotes) * votePrice;
102         }
103         _owner.transfer(0, false, 64);
104     }
```

2.7.2 Function createTokenAccount

- OK

```
228     function createTokenAccount(address tokenRoot) external
            onlyOwner {
229         require(msg.value >= DEFAULT_FEE, Errors.MSG_VALUE_TOO_LOW)
                ;
230         require(!_tokenAccounts.exists(tokenRoot));
231
232         ITokenRoot(tokenRoot).deployEmptyWallet
```

```
233             {value: 0, flag: 64, bounce: true}
234             (tvm.functionId(onTokenWalletDeploy), 0, 0, address(
                    this).value, 1 ton);
235     }
```

2.7.3 Function depositTokens

- OK

```
210     function depositTokens(address tokenRoot) external onlyOwner {
211         require(msg.value >= DEFAULT_FEE, Errors.MSG_VALUE_TOO_LOW)
                ;
212         optional(address) optTokenAccount = _tokenAccounts.fetch(
                tokenRoot);
213         require(optTokenAccount.hasValue(), Errors.
                ACCOUNT_DOES_NOT_EXIST);
214
215         address tokenAccount = optTokenAccount.get();
216
217         ITokenWallet(tokenAccount).getBalance_InternalOwner
218             {value: 0, flag: 64, bounce: true}
219             (tvm.functionId(onTokenWalletGetBalance));
220     }
```

2.7.4 Function depositTons

- OK

```
204     function depositTons(uint128 tons) external onlyOwner {
205         require(msg.value >= tons + 1 ton);
206         _balances[address(0)].total += tons;
207         // _owner.transfer(0, false, 64);
208     }
```

2.7.5 Function onEstimateVotes

- **Major issue: Incorrect computation in `Padawan.onEstimateVotes`**
  The value of `_activeProposalsLength` is wrong if the user sends his votes in multiple batches. Indeed, if this variable measures the size of the mapping `_activeProposals`, it should only be increased in the case `!optActiveProposal.hasValue()`. Otherwise, the value is increased for every batch of votes, and only decreased when all votes have been confirmed/rejected, leading to a over-estimation of the number of entries in the mapping.
- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
60      function onEstimateVotes(
61          uint128 cost,
62          uint128 votePrice,
63          address voteProvider,
64          uint128 votes,
65          bool choice)
66      external override onlyContract {
67          optional(ActiveProposal) optActiveProposal =
                _activeProposals.fetch(msg.sender);
68          ActiveProposal activeProposal = optActiveProposal.hasValue
                () ? optActiveProposal.get() : ActiveProposal(
                voteProvider, votePrice, 0);
69          if(!optActiveProposal.hasValue()) {
70              _activeProposals[msg.sender] = activeProposal;
71          }
72          optional(Balance) optBalance;
73          if(voteProvider == address(0)) {
74              optBalance = _balances.fetch(voteProvider);
75          } else {
76              optional(address) optAccount = _tokenAccounts.fetch(
                    voteProvider);
77              require(optAccount.hasValue(), 115);
78              optBalance = _balances.fetch(optAccount.get());
79          }
80          require(optBalance.hasValue(), 113);
81          require(optBalance.get().total >= (activeProposal.votes *
                votePrice) + cost, 114);
82          _activeProposals[msg.sender].votes += votes;
83          _activeProposalsLength += 1;
84          IProposal(msg.sender).vote
85              {value: 0, flag: 64, bounce: true}
86              (_owner, choice, votes);
87      }
```

2.7.6 Function onTokenWalletDeploy

- **Critical issue: Can empty voting rights in `Padawan.onTokenWalletDeploy`**

  An attacker could send a `onTokenWalletDeploy` message (faking to be a random root token contract) with as argument an existing `voteProvider` of the user, everytime after the user called `depositTokens`. As a result `_balances[account]` is set to 0, emptying the voting rights of the user for that `voteProvider`. Fix: the contract should record the deployment requests and verify that the `msg.sender` is one of them.

```
237     function onTokenWalletDeploy(address account) public {
238         require(!_tokenAccounts.exists(msg.sender), Errors.
                INVALID_CALLER);
239         _tokenAccounts[msg.sender] = account;
240         _balances[account] = Balance(0, 0);
241         _owner.transfer(0, false, 64);
242     }
```

2.7.7 Function onTokenWalletGetBalance

Critical issue: Unbounded voting rights in `Padawan.onTokenWalletGetBalance`

- Because the balance is added to the total (+ =), instead of replacing it, a malicious user could keep calling `depositTokens` to keep increasing his total balance without sending new tokens. Fix: replace + = by =

```
222     function onTokenWalletGetBalance(uint128 balance) public
            onlyContract {
223         optional(Balance) optBalance = _balances.fetch(msg.sender);
224         require(optBalance.hasValue(), Errors.
                NOT_AUTHORIZED_CONTRACT);
225         _balances[msg.sender].total += balance;
226     }
```

2.7.8 Function reclaimDeposit

Critical issue: Race condition in `Padawan.reclaimDeposit`

- Because `locked` is only increased in `Padawan.confirmVote`, a malicious user could `reclaimDeposit` just after `Padawan.onEstimateVotes` and before `Padawan.confirmVote`. In this case, the user can empty his balance, while still participating to the vote. Slashing will not be possible later if his vote was incorrect. Fix: locked amount should be recomputed for every `reclaimDeposit` from all the active proposals.

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
118     function reclaimDeposit(address voteProvider, uint128 amount,
            address returnTo) external onlyOwner {
119         require(_reclaim.amount == 0, 130);
120         require(msg.value >= QUERY_STATUS_FEE *
                _activeProposalsLength + 1 ton, Errors.
                MSG_VALUE_TOO_LOW);
121         address balanceProvider = address(0);
122         if(voteProvider != address(0)) {
123             optional(address) optAccount = _tokenAccounts.fetch(
                    voteProvider);
124             require(optAccount.hasValue(), 117);
125             balanceProvider = optAccount.get();
126         }
127         optional(Balance) optBalance = _balances.fetch(
                balanceProvider);
128         require(optBalance.hasValue(), 131);
129         Balance balance = optBalance.get();
130         require(amount <= balance.total, Errors.NOT_ENOUGH_VOTES);
131         require(returnTo != address(0), 132);
132
133         _reclaim = Reclaim(balanceProvider, amount, returnTo);
134
```

```
135         if (amount <= balance.total - balance.locked) {
136             _doReclaim();
137         }
138
139         optional(address, ActiveProposal) optActiveProposal =
                _activeProposals.min();
140         while (optActiveProposal.hasValue()) {
141             (address addrActiveProposal,) = optActiveProposal.get()
                    ;
142             IProposal(addrActiveProposal).queryStatus
143                 {value: QUERY_STATUS_FEE, bounce: true, flag: 1}
144                 ();
145             optActiveProposal = _activeProposals.next(
                    addrActiveProposal);
146         }
147     }
```

2.7.9 Function rejectVote

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
106     function rejectVote(uint128 votes, uint16 errorCode) external
            onlyContract { votes; errorCode;
107         optional(ActiveProposal) optActiveProposal =
                _activeProposals.fetch(msg.sender);
108         require(optActiveProposal.hasValue(), 112);
109         ActiveProposal activeProposal = optActiveProposal.get();
110         activeProposal.votes -= votes;
111         if (activeProposal.votes == 0) {
112             delete _activeProposals[msg.sender];
113             _activeProposalsLength -= 1;
114         }
115         _owner.transfer(0, false, 64);
116     }
```

2.7.10 Function updateStatus

- Minor issue (readability): the test for recomputation of locked amount should be `==` instead of `<=` as the former locked amount can never be strictly smaller than a given proposal cost.
- Minor issue (readability): the recomputation of the locked amount should be moved to an internal function, and reused in `reclaimDeposit` to avoid the race condition with `confirmVote`
- Minor issue (code repetition): `delete _activeProposals[msg.sender]` is in both clauses of the `if` and could be moved outside.
- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
149     function updateStatus(ProposalState state) external
            onlyContract {
150         optional(ActiveProposal) optActiveProposal =
                _activeProposals.fetch(msg.sender);
151         require(optActiveProposal.hasValue());
152         ActiveProposal activeProposal = optActiveProposal.get();
153
154         if (state >= ProposalState.Ended) {
155             address balanceProvider = address(0);
156             if(activeProposal.voteProvider != address(0)) {
157                 optional(address) optAccount = _tokenAccounts.fetch
                        (activeProposal.voteProvider);
158                 require(optAccount.hasValue(), 117);
159                 balanceProvider = optAccount.get();
160             }
161             Balance balance = _balances[balanceProvider];
162             if(balance.locked <= activeProposal.votes *
                    activeProposal.votePrice) {
163                 delete _activeProposals[msg.sender];
164                 uint128 max;
165                 optional(address, ActiveProposal)
                        optActiveProposal2 = _activeProposals.min();
166                 while (optActiveProposal2.hasValue()) {
167                     (address addrActiveProposal, ActiveProposal
                            activeProposal2) = optActiveProposal2.get()
                            ;
168                     if(activeProposal2.votes * activeProposal2.
                            votePrice > max && activeProposal2.
                            voteProvider == activeProposal.voteProvider
                            ) {
169                         max = activeProposal2.votes *
                                activeProposal2.votePrice;
170                     }
171                     optActiveProposal2 = _activeProposals.next(
                            addrActiveProposal);
172                 }
173                 _balances[balanceProvider].locked = max;
174             } else {
175                 delete _activeProposals[msg.sender];
176             }
177             _activeProposalsLength -= 1;
178             if(_reclaim.amount != 0) {
179                 balance = _balances[_reclaim.balanceProvider];
180                 if (_reclaim.amount <= balance.total - balance.
                        locked) {
181                     _doReclaim();
182                 }
183             }
184         }
185     }
```

2.7.11 Function vote

- OK

```
53      function vote(address proposal, bool choice, uint128 votes)
            external onlyOwner {
54          require(msg.value >= VOTE_FEE, Errors.MSG_VALUE_TOO_LOW);
55          IProposal(proposal).estimateVotes
56              {value: 0, flag: 64, bounce: true}
57              (votes, choice);
58      }
```

2.8 Internal Method Definitions

2.8.1 Function _doReclaim

- OK

```
191     function _doReclaim() private inline {
192         if(_reclaim.balanceProvider == address(0)) {
193             _reclaim.returnTo.transfer(_reclaim.amount, true, 1);
194         } else {
195             ITokenWallet(_reclaim.balanceProvider).transfer
196                 {value: 0.2 ton} // refactor
197                 (_reclaim.returnTo, _reclaim.amount, 0.1 ton);
198         }
199         _balances[_reclaim.balanceProvider].total -= _reclaim.
                amount;
200         delete _reclaim;
201         _owner.transfer(0, false, 64);
202     }
```

Chapter 3

Contract Proposal

Contents

3.1 Overview

In file `Proposal.sol`

3.2 Contract Inheritance

Base	
PadawanResolver	
GroupResolver	
IProposal	
IGroupCallback	

3.3 Static Variable Definitions

- OK

```
15    address static _deployer;

16    uint32 static _id;
```

3.4 Variable Definitions

- OK

```
18    address _client;

20    uint128 _votePrice;

21    uint128 _voteTotal;

22    address _voteProvider;

24    address[] _whiteList;

25    bool _openProposal = false;

27    ProposalInfo _proposalInfo;

29    ProposalResults _results;

30    VoteCountModel _voteCountModel;
```

3.5 Constructor Definitions

3.5.1 Constructor

- Minor issue: there is a limitation to 16 kB for deploy messages. For this constructor, the deploy message contains the code of `Proposal`, the title and the code of `Padawan`. Thus, it might become a problem in the future. There is already a mechanism in the infrastructure to download codes from the `DemiurgeStore`, this contract should take advantage of it.
- Minor issue: the `_voteCountModel` variable is initialized to `SoftMajority` in this constructor, but it is not used anywhere. Consider removing it if no future use.

```
32      constructor(
33          address client,
34          string title,
35          uint128 votePrice,
36          uint128 voteTotal,
37          address voteProvider,
38          address group,
39          address[] whiteList,
40          string proposalType,
41          TvmCell specific,
42          TvmCell codePadawan
43      ) public {
44          require(_deployer == msg.sender);
45
46          _client = client;
47
48          _votePrice = votePrice;
49          _voteTotal = voteTotal;
50          _voteProvider = voteProvider;
51
52          _proposalInfo.title = title;
53          _proposalInfo.start = uint32(now);
54          _proposalInfo.end = uint32(now + 60 * 60 * 24 * 7);
55          _proposalInfo.proposalType = proposalType;
56          _proposalInfo.specific = specific;
57          _proposalInfo.state = ProposalState.New;
58          _proposalInfo.totalVotes = voteTotal;
59
60          _codePadawan = codePadawan;
61
62          if(group != address(0)) {
63              _getGroupMembers(group);
64          } else if (!whiteList.empty()) {
65              _whiteList = whiteList;
66          } else  {
67              _openProposal = true;
68          }
69
70          _voteCountModel = VoteCountModel.SoftMajority;
71      }
```

3.6 Public Method Definitions

3.6.1 Function estimateVotes

- OK

```
78      function estimateVotes(uint128 votes, bool choice) external
            override {
79          IEstimateVotesCallback(msg.sender).onEstimateVotes
80              {value: 0, flag: 64, bounce: true}
81              (votes * _votePrice, _votePrice, _voteProvider, votes,
                    choice);
82      }
```

3.6.2 Function getAll

- OK

```
199     function getAll() public view override returns (ProposalInfo
            info) {
200         info = _proposalInfo;
201     }
```

3.6.3 Function getCurrentVotes

- OK

```
212     function getCurrentVotes() external override view returns (
            uint128 votesFor, uint128 votesAgainst) {
213         return (_proposalInfo.votesFor, _proposalInfo.votesAgainst)
                ;
214     }
```

3.6.4 Function getInfo

- OK

```
208     function getInfo() public view returns (ProposalInfo info) {
209         info = _proposalInfo;
210     }
```

3.6.5 Function getVotingResults

- OK

```
203     function getVotingResults() public view returns (
            ProposalResults vr) {
204         require(_proposalInfo.state > ProposalState.Ended, Errors.
                VOTING_HAS_NOT_ENDED);
205         vr = _results;
206     }
```

3.6.6 Function onGetMembers

- **Critical issue: No permission check on `Proposal.onGetMembers`**
  No check is performed on the sender of `onGetMembers`. An attacker could use it to fill the `_whiteList` variable with malicious members.

```
220     function onGetMembers(string name, address[] members) public
            override onlyContract { name;
221         _whiteList = members;
222     }
```

3.6.7 Function queryStatus

- Minor issue: a `require` should check that the message contains enough value to send the message.

```
191     function queryStatus() external override {
192         IPadawan(msg.sender).updateStatus
193             {value: 0, flag: 64, bounce: true}
194             (_proposalInfo.state);
195     }
```

3.6.8 Function vote

- Minor issue: a `require` should check that the message contains enough value to send back the reply;
- Minor issue: given that the constructor initializes `_proposalInfo.start` to `now`, it is impossible for this function to return the `VOTING_NOT_STARTED` error.
- Minor issue: the transaction could be aborted if a `onProposalPassed` message is sent by `_finalize` (in `_wrapUp`), together with `rejectVote` or `confirmVote` messages, because of the flag 64. Need to test what happens if two messages are sent by the same transaction, with one of them containing the flag 64.

```
84      function vote(address padawanOwner, bool choice, uint128 votes)
             external override {
85          address addrPadawan = resolvePadawan(padawanOwner);
86          uint16 errorCode = 0;
87
88          require(_openProposal || _findInWhiteList(padawanOwner),
                Errors.INVALID_CALLER);
89
90          if (addrPadawan != msg.sender) {
91              errorCode = Errors.NOT_AUTHORIZED_CONTRACT;
92          } else if (now < _proposalInfo.start) {
93              errorCode = Errors.VOTING_NOT_STARTED;
94          } else if (now > _proposalInfo.end) {
```

```
95              errorCode = Errors.VOTING_HAS_ENDED;
96          }
97
98          if (errorCode > 0) {
99              IPadawan(msg.sender).rejectVote{value: 0, flag: 64,
                    bounce: true}(votes, errorCode);
100         } else {
101             IPadawan(msg.sender).confirmVote{value: 0, flag: 64,
                    bounce: true}(votes, _votePrice, _voteProvider);
102             if (choice) {
103                 _proposalInfo.votesFor += votes;
104             } else {
105                 _proposalInfo.votesAgainst += votes;
106             }
107         }
108
109         _wrapUp();
110     }
```

3.6.9 Function wrapUp

- OK

```
73      function wrapUp() external override {
74          _wrapUp();
75          msg.sender.transfer(0, false, 64);
76      }
```

3.7 Internal Method Definitions

3.7.1 Function _buildPadawanState

- OK

```
183     function _buildPadawanState(address owner) internal view
            override returns (TvmCell) {
184         return tvm.buildStateInit({
185             contr: Padawan,
186             varInit: {_deployer: _deployer, _owner: owner},
187             code: _codePadawan
188         });
189     }
```

3.7.2 Function _calculateVotes

- OK

```
161     function _calculateVotes(
162         uint128 yes,
163         uint128 no
```

```
164     ) private view returns (bool) {
165         bool passed = false;
166         passed = _softMajority(yes, no);
167         return passed;
168     }
```

3.7.3 Function _changeState

- OK

```
179     function _changeState(ProposalState state) private inline {
180         _proposalInfo.state = state;
181     }
```

3.7.4 Function _finalize

- OK

```
112     function _finalize(bool passed) private {
113         _results = ProposalResults(
114             uint32(0),
115             passed,
116             _proposalInfo.votesFor,
117             _proposalInfo.votesAgainst,
118             _voteTotal,
119             _voteCountModel,
120             uint32(now)
121         );
122 
123         ProposalState state = passed ? ProposalState.Passed :
                ProposalState.NotPassed;
124 
125         _changeState(state);
126 
127         IClient(address(_client)).onProposalPassed{value: 1 ton} (
                _proposalInfo);
128     }
```

3.7.5 Function _findInWhiteList

- OK

```
224     function _findInWhiteList(address padawanOwner) view private
            returns (bool) {
225         for(uint32 index = 0; index < _whiteList.length; index++) {
226             if(_whiteList[index] == padawanOwner) {
227                 return true;
228             }
229         }
230         return false;
231     }
```

3.7.6 Function _getGroupMembers

- OK

```
233     function _getGroupMembers(address group) view private {
234         IGroup(group).getMembers();
235     }
```

3.7.7 Function _softMajority

- **Critical issue: Division by 0 in `Proposal._softMajority`**
  If `totalVotes=1`, this function fails with division by 0. Fix: the function should check that `totalVotes>1`, and add special cases for `totalVotes=1` and `totalVotes=0`
- Minor issue (readability): use `returns (bool passed)` to avoid the need to define a temporary variable and to return it.

```
170     function _softMajority(
171         uint128 yes,
172         uint128 no
173     ) private view returns (bool) {
174         bool passed = false;
175         passed = yes >= 1 + (_voteTotal / 10) + (no * ((_voteTotal
                / 2) - (_voteTotal / 10))) / (_voteTotal / 2);
176         return passed;
177     }
```

3.7.8 Function _tryEarlyComplete

- Minor issue (readability): use `returns (bool completed, bool passed)` to avoid the need to define temporary variables and to return them.

```
130     function _tryEarlyComplete(
131         uint128 yes,
132         uint128 no
133     ) private view returns (bool, bool) {
134         (bool completed, bool passed) = (false, false);
135         if (yes * 2 > _voteTotal) {
136             completed = true;
137             passed = true;
138         } else if(no * 2 >= _voteTotal) {
139             completed = true;
140             passed = false;
141         }
142         return (completed, passed);
143     }
```

3.7.9 Function _wrapUp

- Minor issue: the function could immediately check if the state is above `Ended` to avoid recomputing again when the state cannot change anymore;
- Minor issue: there is no need to call `_changeState` before calling `_finalize`, as `_finalize` always calls `_changeState` and will thus override the state written in this function;

```
145     function _wrapUp() private {
146         (bool completed, bool passed) = (false, false);
147 
148         if (now > _proposalInfo.end) {
149             completed = true;
150             passed = _calculateVotes(_proposalInfo.votesFor,
                    _proposalInfo.votesAgainst);
151         } else {
152             (completed, passed) = _tryEarlyComplete(_proposalInfo.
                    votesFor, _proposalInfo.votesAgainst);
153         }
154 
155         if (completed) {
156             _changeState(ProposalState.Ended);
157             _finalize(passed);
158         }
159     }
```

Chapter 4

Contract Group

Contents

4.1 Overview

In file `Group.sol`

4.2 Contract Inheritance

Base	
IGroup	

4.3 Static Variable Definitions

```
11      string static _name;
```

4.4 Variable Definitions

```
12      address[] _members;
```

4.5 Constructor Definitions

4.5.1 Constructor

- **Critical issue: No permission check in `Group.constructor`**
  No permission check is performed on the deployer of the contract. As a consequence, an attacker could deploy a `Group` contract for a given name before the user, if it can predict that the user will use that name, and the attacker could initialize the contract with his own list of (malicious) members. Fix: add a static variable in the contract, with the only allowed deployer of the contract and check that the sender is the allowed deployer in the constructor.

```
15      constructor(address[] initialMembers) public onlyContract {
16          _members = initialMembers;
17      }
```

4.6 Public Method Definitions

4.6.1 Function addMember

- **Critical issue: No permission check in `Group.addMember`**
  An attacker could add any member to the group because no permission check is performed in this function
- Minor issue: a member can be added several times in the group. Fix: use a mapping and only add non-existing members.
- Minor issue: the argument `idProposal` is not used.

```
25      function addMember(uint128 idProposal, address member) public
            onlyContract {
26          idProposal;
27          _members.push(member);
28      }
```

4.6.2 Function getMembers

- OK

```
19      function getMembers() override public onlyContract {
20          IGroupCallback(msg.sender).onGetMembers
21              {value: 0, flag: 64, bounce: true}
22              (_name, _members);
23      }
```

4.6.3 Function removeMember

- **Critical issue: No permission check on `removeMember`**
  An attacker could remove any member of the group, as no permission check is performed.
- Minor issue: the argument `idProposal` is not used.

```
30      function removeMember(uint128 idProposal, address member)
            public onlyContract {
31          idProposal;
32          address[] members;
33          for(uint32 index = 0; index < _members.length; index++) {
34              if(_members[index] != member) {
35                  members.push(_members[index]);
36              }
37          }
38          _members = members;
39      }
```

Chapter 5

Contract SmvRoot

Contents

5.1 Overview

In file `SmvRoot.sol`

5.2 Contract Inheritance

Base	
ISmvRoot	
ISmvRootStoreCallback	
PadawanResolver	
ProposalResolver	
GroupResolver	
ProposalFactoryResolver	
Checks	

5.3 Constant Definitions

- OK

```
36    uint8 constant CHECK_PROPOSAL = 1;
37    uint8 constant CHECK_PADAWAN = 2;
38    uint8 constant CHECK_GROUP = 4;
39    uint8 constant CHECK_PROPOSAL_FACTORY = 8;
40    uint8 constant CHECK_BFTG_ROOT_ADDRESS = 16;
```

5.4 Variable Definitions

```
55    address _addrSmvRootStore;
56    address _addrBftgRoot;
57    address _addrProposalFactory;
91    bool public _inited = false;
132   uint32 _deployedPadawansCounter;
141   uint32 _deployedProposalsCounter;
```

5.5 Modifier Definitions

5.5.1 Modifier onlyStore

- OK

```
59    modifier onlyStore() {
60        require(msg.sender == _addrSmvRootStore, Errors.ONLY_STORE)
              ;
61        _;
62    }
```

5.6 Constructor Definitions

5.6.1 Constructor

- **Critical issue: Administrative Take-over in `SmvRoot.constructor`**
  No test is performed to verify the sender in the case `msg.sender != address(0)`. An attacker could use it to deploy the contract himself for another user, providing its own `addrSmvRootStore`, i.e. with his own code for most contracts.

```
64      constructor(address addrSmvRootStore) public {
65          if (msg.sender == address(0)) {
66              require(msg.pubkey() == tvm.pubkey(), Errors.
                    ONLY_SIGNED);
67          }
68          require(addrSmvRootStore != address(0), Errors.
                STORE_UNDEFINED);
69          tvm.accept();
70
71          _addrSmvRootStore = addrSmvRootStore;
72          ISmvRootStore(_addrSmvRootStore).queryCode
73              {value: 0.2 ton, bounce: true}
74              (ContractCode.Proposal);
75          ISmvRootStore(_addrSmvRootStore).queryCode
76              {value: 0.2 ton, bounce: true}
77              (ContractCode.Padawan);
78          ISmvRootStore(_addrSmvRootStore).queryCode
79              {value: 0.2 ton, bounce: true}
80              (ContractCode.Group);
81          ISmvRootStore(_addrSmvRootStore).queryCode
82              {value: 0.2 ton, bounce: true}
83              (ContractCode.ProposalFactory);
84          ISmvRootStore(_addrSmvRootStore).queryAddr
85              {value: 0.2 ton, bounce: true}
86              (ContractAddr.BftgRoot);
87
88          _createChecks();
89      }
```

5.7 Public Method Definitions

5.7.1 Function _deployProposal

- Minor issue: the reason to make this function public instead of internal is unclear.
- Minor issue: this function should check that the code of `Proposal` is ready before further processing.

```
194     function _deployProposal(
195         address client,
196         string title,
197         uint128 votePrice,
198         uint128 voteTotal,
199         address voteProvider,
200         address group,
201         address[] whiteList,
202         string proposalType,
203         TvmCell specific
204     ) public onlyMe {
205         TvmCell state = _buildProposalState(
                _deployedProposalsCounter);
206         new Proposal {stateInit: state, value: START_BALANCE}(
207             client,
208             title,
209             votePrice,
210             voteTotal,
211             voteProvider,
212             group,
213             whiteList,
214             proposalType,
215             specific,
216             _codePadawan
217         );
218         _deployedProposalsCounter++;
219     }
```

5.7.2 Function deployGroup

- **Major issue: No check on `SmvRoot.deployGroup`**
  No check is performed to verify that the group does not already exist. As a consequence, the use might believe it created a group with the list of members given in argument, whereas in fact, the group already existed and contains another set of members. Fix: this function could check for the existence using bounced messages, and use a callback to return the result to the caller.

```
221     function deployGroup(string name, address[] initialMembers)
            public onlyContract {
222         TvmCell state = _buildGroupState(name);
223         new Group
224             {stateInit: state, value: START_BALANCE}
225             (initialMembers);
226     }
```

5.7.3 Function deployPadawan

- Minor issue: the function should check that the code of the Padawan contract was correctly initialized.

```
134     function deployPadawan(address owner) external onlyContract {
135         require(msg.value >= DEPLOY_FEE);
136         require(owner != address(0));
137         TvmCell state = _buildPadawanState(owner);
138         new Padawan{stateInit: state, value: START_BALANCE + 2 ton
                }();
139     }
```

5.7.4 Function deployProposal

- Minor issue: the function should check that the code of the Proposal contract was correctly initialized.

```
143     function deployProposal(
144         address client,
145         string title,
146         uint128 votePrice,
147         uint128 voteTotal,
148         address voteProvider,
149         address group,
150         address[] whiteList,
151         string proposalType,
152         TvmCell specific
153     ) external override onlyContract {
154         require(msg.sender == _addrProposalFactory);
155         require(msg.value >= DEPLOY_PROPOSAL_FEE);
156         TvmBuilder b;
157         b.store(specific);
158         TvmCell cellSpecific = b.toCell();
159         _beforeProposalDeploy(
160             client,
161             title,
162             votePrice,
163             voteTotal,
164             voteProvider,
165             group,
166             whiteList,
167             proposalType,
168             cellSpecific
169         );
170     }
```

5.7.5 Function getStats

- OK

```
246     function getStats() public view returns (uint32
            deployedPadawansCounter, uint32 deployedProposalsCounter) {
247         deployedPadawansCounter = _deployedPadawansCounter;
248         deployedProposalsCounter = _deployedProposalsCounter;
249     }
```

5.7.6 Function getStored

- OK

```
230     function getStored() public view returns (
231         TvmCell codePadawan,
232         TvmCell codeProposal,
233         TvmCell codeGroup,
234         TvmCell codeProposalFactory,
235         address addrBftgRoot,
236         address proposalFactory
237     ) {
238         codePadawan = _codePadawan;
239         codeProposal = _codeProposal;
240         codeGroup = _codeGroup;
241         codeProposalFactory = _codeProposalFactory;
242         addrBftgRoot = _addrBftgRoot;
243         proposalFactory = _addrProposalFactory;
244     }
```

5.7.7 Function updateAddr

- OK

```
123     function updateAddr(ContractAddr kind, address addr) external
            override onlyStore {
124         require(addr != address(0));
125         if (kind == ContractAddr.BftgRoot) {
126             _addrBftgRoot = addr;
127             _passCheck(CHECK_BFTG_ROOT_ADDRESS);
128         }
129         _onInit();
130     }
```

5.7.8 Function updateCode

- OK

```
103     function updateCode(
104         ContractCode kind,
105         TvmCell code
106     ) external override onlyStore {
107         if (kind == ContractCode.Proposal) {
108             _codeProposal = code;
109             _passCheck(CHECK_PROPOSAL);
110         } else if (kind == ContractCode.Padawan) {
111             _codePadawan = code;
112             _passCheck(CHECK_PADAWAN);
113         } else if (kind == ContractCode.Group) {
114             _codeGroup = code;
115             _passCheck(CHECK_GROUP);
116         } else if (kind == ContractCode.ProposalFactory) {
117             _codeProposalFactory = code;
```

```
118             _passCheck(CHECK_PROPOSAL_FACTORY);
119         }
120         _onInit();
121     }
```

5.8 Internal Method Definitions

5.8.1 Function _beforeProposalDeploy

- Minor issue: if there is no future use for `proposal`, this function should be replaced by a direct call to `_deployProposal`.

```
172     function _beforeProposalDeploy(
173         address client,
174         string title,
175         uint128 votePrice,
176         uint128 voteTotal,
177         address voteProvider,
178         address group,
179         address[] whiteList,
180         string proposalType,
181         TvmCell specific
182     ) private view {
183         TvmCell state = _buildProposalState(
                _deployedProposalsCounter);
184         uint256 hashState = tvm.hash(state);
185         address proposal = address.makeAddrStd(0, hashState);
186         // IClient(_addrDensRoot).onProposalDeploy
187         //     {value: 1 ton, bounce: true}
188         //     (proposal, proposalType, specific);
189         this._deployProposal
190             {value: 4 ton}
191             (client, title, votePrice, voteTotal, voteProvider,
                    group, whiteList, proposalType, specific);
192     }
```

5.8.2 Function _createChecks

- OK

```
42     function _createChecks() private inline {
43         _checkList =
44             CHECK_PROPOSAL |
45             CHECK_PADAWAN |
46             CHECK_GROUP |
47             CHECK_PROPOSAL_FACTORY |
48             CHECK_BFTG_ROOT_ADDRESS;
49     }
```

5.8.3 Function _onInit

- OK

```
93      function _onInit() private {
94          if(_isCheckListEmpty() && !_inited) {
95              _inited = true;
96              TvmCell state = _buildProposalFactoryState(address(this
                    ));
97              _addrProposalFactory = new ProposalFactory
98                  {stateInit: state, value: 0.2 ton}
99                  ();
100         }
101     }
```

Chapter 6

Contract ProposalFactory

Contents

6.1 Overview

In file `ProposalFactory.sol`

6.2 Contract Inheritance

Base	

6.3 Static Variable Definitions

- OK

```
14      address static _deployer;
```

6.4 Constructor Definitions

6.4.1 Constructor

- OK

```
16      constructor() public onlyContract {
17          require(_deployer == msg.sender, Errors.ONLY_DEPLOYER);
18      }
```

6.5 Public Method Definitions

6.5.1 Function deployAddMemberProposal

- OK

```
75      function deployAddMemberProposal(
76          address client,
77          string title,
78          uint128 votePrice,
79          uint128 voteTotal,
80          address voteProvider,
81          address group,
82          address[] whiteList,
83          AddMemberProposalSpecific specific
84      ) external view onlyContract {
85          require(msg.value >= DEPLOY_PROPOSAL_PAY + 1 ton);
86          TvmBuilder b;
87          b.store(specific);
88          TvmCell cellSpecific = b.toCell();
89          ISmvRoot(_deployer).deployProposal
90              {value: 0, flag: 64, bounce: true}
91              (
92                  client,
93                  title,
94                  votePrice,
95                  voteTotal,
96                  voteProvider,
97                  group,
98                  whiteList,
99                  'add-member',
100                 cellSpecific
101             );
102     }
```

6.5.2 Function deployContestProposal

- OK

```
20      function deployContestProposal(
21          address client,
22          string title,
23          address group,
24          ContestProposalSpecific specific
25      ) external view onlyContract {
26          require(msg.value >= DEPLOY_PROPOSAL_PAY + 1 ton);
27          TvmBuilder b;
28          b.store(specific);
29          TvmCell cellSpecific = b.toCell();
30          address[] arr;
31          ISmvRoot(_deployer).deployProposal
32              {value: 0, flag: 64, bounce: true}
33              (
34                  client,
35                  title,
36                  1 ton,
37                  1000000000,
38                  address(0),
39                  group,
40                  arr,
41                  'contest',
42                  cellSpecific
43              );
44      }
```

6.5.3 Function deployRemoveMemberProposal

- OK

```
46      function deployRemoveMemberProposal(
47          address client,
48          string title,
49          uint128 votePrice,
50          uint128 voteTotal,
51          address voteProvider,
52          address group,
53          address[] whiteList,
54          RemoveMemberProposalSpecific specific
55      ) external view onlyContract {
56          require(msg.value >= DEPLOY_PROPOSAL_PAY + 1 ton);
57          TvmBuilder b;
58          b.store(specific);
59          TvmCell cellSpecific = b.toCell();
60          ISmvRoot(_deployer).deployProposal
61              {value: 0, flag: 64, bounce: true}
62              (
63                  client,
64                  title,
65                  votePrice,
66                  voteTotal,
67                  voteProvider,
68                  group,
69                  whiteList,
70                  'remove-member',
71                  cellSpecific
```

```
72                 );
73     }
```

Chapter 7

Contract PadawanResolver

Contents

7.1 Overview

In file `PadawanResolver.sol`

7.2 Variable Definitions

```
8     TvmCell _codePadawan;
```

7.3 Public Method Definitions

7.3.1 Function resolvePadawan

- OK

```
10    function resolvePadawan(address owner) public view returns (
          address addrPadawan) {
11        TvmCell state = _buildPadawanState(owner);
12        uint256 hashState = tvm.hash(state);
13        addrPadawan = address.makeAddrStd(0, hashState);
14    }
```

7.4 Internal Method Definitions

7.4.1 Function _buildPadawanState

- Minor issue: this function should fail (`require`) if the `_codeJuryGroup` variable has not yet been initialized. A global boolean could be used for that, set in an internal function initializing both global variables.

```
16      function _buildPadawanState(address owner) internal virtual
            view returns (TvmCell) {
17          return tvm.buildStateInit({
18              contr: Padawan,
19              varInit: {_deployer: address(this), _owner: owner},
20              code: _codePadawan
21          });
22      }
```

Chapter 8

Contract ProposalFactoryResolver

Contents

8.1 Overview

In file `ProposalFactoryResolver.sol`

8.2 Variable Definitions

```
6     TvmCell _codeProposalFactory;
```

8.3 Public Method Definitions

8.3.1 Function resolveProposalFactory

- OK

```
8     function resolveProposalFactory(address deployer) public view
          returns (address addrProposalFactory) {
9         TvmCell state = _buildProposalFactoryState(deployer);
```

```
10          uint256 hashState = tvm.hash(state);
11          addrProposalFactory = address.makeAddrStd(0, hashState);
12      }
```

8.4 Internal Method Definitions

8.4.1 Function _buildProposalFactoryState

- Minor issue: this function should fail (`require`) if the `_codeProposalFactory` variable has not yet been initialized. A global boolean could be used for that, set in an internal function initializing both global variables.

```
14      function _buildProposalFactoryState(address deployer) internal
            view returns (TvmCell) {
15          return tvm.buildStateInit({
16              contr: ProposalFactory,
17              varInit: {_deployer: deployer},
18              code: _codeProposalFactory
19          });
20      }
```

Chapter 9

Contract ProposalResolver

Contents

9.1 Overview

In file `ProposalResolver.sol`

9.2 Variable Definitions

```
6     TvmCell _codeProposal;
```

9.3 Public Method Definitions

9.3.1 Function resolveProposal

- OK

```
8     function resolveProposal(uint32 id) public view returns (
          address addrProposal) {
9         TvmCell state = _buildProposalState(id);
10        uint256 hashState = tvm.hash(state);
11        addrProposal = address.makeAddrStd(0, hashState);
12    }
```

9.4 Internal Method Definitions

9.4.1 Function _buildProposalState

- Minor issue: this function should fail (`require`) if the `_codeProposalFactory` variable has not yet been initialized. A global boolean could be used for that, set in an internal function initializing both global variables.

```
14      function _buildProposalState(uint32 id) internal view returns (
            TvmCell) {
15          return tvm.buildStateInit({
16              contr: Proposal,
17              varInit: {_deployer: address(this), _id: id},
18              code: _codeProposal
19          });
20      }
```

Chapter 10

Contract GroupResolver

Contents

10.1 Overview

In file `GroupResolver.sol`

10.2 Variable Definitions

TvmCell	_codeGroup	
		used in @16.GroupResolver._buildGroupState

```
8     TvmCell _codeGroup;
```

10.3 Public Method Definitions

10.3.1 Function resolveGroup

- OK

```
10      function resolveGroup(string name) public view returns (address
            group) {
11          TvmCell state = _buildGroupState(name);
12          uint256 hashState = tvm.hash(state);
13          group = address.makeAddrStd(0, hashState);
14      }
```

10.4 Internal Method Definitions

10.4.1 Function _buildGroupState

- Minor issue: this function should fail (`require`) if the `_codeGroup` variable has not yet been initialized. A global boolean could be used for that, set in an internal function initializing both global variables.

```
16      function _buildGroupState(string name) internal virtual view
            returns (TvmCell) {
17          return tvm.buildStateInit({
18              contr: Group,
19              varInit: {_name: name},
20              code: _codeGroup
21          });
22      }
```

Chapter 11

Contract SmvRootStore

Contents

11.1 Overview

In file `SmvRootStore.sol`

11.2 General Minor-level Remarks

In general, the infrastructure would be safer if this contract would be implemented in two phases:

- In the Initialization phase, the contract is waiting for all the `setXXX` methods to be called to initialize all the fields. A bitmap can be used to keep the current initialization state. Any attempt to user a `getXXX` method should fail.

- In the Post-Initalization phase, the contract accepts to reply to `getXXX` methods, but `setXXX` methods are disabled.

There is also an inconsistency between the getters and setters: getters are generic (they take a `kind` as argument), whereas setters are specific (there is a different one for every kind).

11.3 Contract Inheritance

Base	
ISmvRootStore	

11.4 Variable Definitions

```
10    mapping(uint8 => address) public _addrs;

11    mapping(uint8 => TvmCell) public _codes;
```

11.5 Public Method Definitions

11.5.1 Function queryAddr

- Minor issue: a `require` could be added to fail if `kind` is not a well-known kind.

```
36    function queryAddr(ContractAddr kind) external override {
37        address addr = _addrs[uint8(kind)];
38        ISmvRootStoreCallback(msg.sender).updateAddr{value: 0, flag
              : 64, bounce: false}(kind, addr);
39    }
```

11.5.2 Function queryCode

- Minor issue: a `require` could be added to fail if `kind` is not a well-known kind.

```
31    function queryCode(ContractCode kind) external override {
32        TvmCell code = _codes[uint8(kind)];
33        ISmvRootStoreCallback(msg.sender).updateCode{value: 0, flag
              : 64, bounce: false}(kind, code);
34    }
```

11.5.3 Function setBftgRootAddr

- OK

```
26      function setBftgRootAddr(address addr) public override signed {
27          require(addr != address(0));
28          _addrs[uint8(ContractAddr.BftgRoot)] = addr;
29      }
```

11.5.4 Function setGroupCode

- Minor issue: the infrastructure would probably be safer if the expected code hash is hardcoded in the source code, and check through a `require`

```
19      function setGroupCode(TvmCell code) public override signed {
20          _codes[uint8(ContractCode.Group)] = code;
21      }
```

11.5.5 Function setPadawanCode

- Minor issue: the infrastructure would probably be safer if the expected code hash is hardcoded in the source code, and check through a `require`

```
13      function setPadawanCode(TvmCell code) public override signed {
14          _codes[uint8(ContractCode.Padawan)] = code;
15      }
```

11.5.6 Function setProposalCode

- Minor issue: the infrastructure would probably be safer if the expected code hash is hardcoded in the source code, and check through a `require`

```
16      function setProposalCode(TvmCell code) public override signed {
17          _codes[uint8(ContractCode.Proposal)] = code;
18      }
```

11.5.7 Function setProposalFactoryCode

- Minor issue: the infrastructure would probably be safer if the expected code hash is hardcoded in the source code, and check through a `require`

```
22      function setProposalFactoryCode(TvmCell code) public override
            signed {
23          _codes[uint8(ContractCode.ProposalFactory)] = code;
24      }
```

Chapter 12

Contract Base

Contents

12.1 Overview

In file `Base.sol`

12.2 Constant Definitions

```
8     uint64 constant DEFAULT_FEE          = 1 ton;

10    uint16 constant ERROR_DIFFERENT_CALLER =  211;

12    uint64 constant START_BALANCE        = 3 ton;

13    uint64 constant DEPLOYER_FEE         = 0.1 ton;

14    uint64 constant PROCESS_FEE          = 0.3 ton;

15    uint64 constant VOTE_FEE             = 1 ton;

16    uint64 constant DEPLOY_FEE           = START_BALANCE +
          DEPLOYER_FEE;
```

```
17      uint64 constant DEPLOY_PAY          = DEPLOY_FEE + PROCESS_FEE;
18      uint64 constant DEPLOY_PROPOSAL_FEE = 3 ton;
19      uint64 constant DEPLOY_PROPOSAL_PAY = DEPLOY_PROPOSAL_FEE +
            PROCESS_FEE;
20      uint64 constant DEPOSIT_TONS_FEE    = 1 ton;
21      uint64 constant DEPOSIT_TONS_PAY    = DEPOSIT_TONS_FEE +
            PROCESS_FEE;
22      uint64 constant DEPOSIT_TOKENS_FEE  = 0.5 ton +
            DEPOSIT_TONS_FEE;
23      uint64 constant DEPOSIT_TOKENS_PAY  = DEPOSIT_TOKENS_FEE +
            PROCESS_FEE;
24      uint64 constant TOKEN_ACCOUNT_FEE   = 2 ton;
25      uint64 constant TOKEN_ACCOUNT_PAY   = TOKEN_ACCOUNT_FEE +
            PROCESS_FEE;
26      uint64 constant QUERY_STATUS_FEE    = 0.2 ton;
27      uint64 constant QUERY_STATUS_PAY    = QUERY_STATUS_FEE +
            DEF_RESPONSE_VALUE;
29      uint64 constant DEF_RESPONSE_VALUE = 0.03 ton;
30      uint64 constant DEF_COMPUTE_VALUE = 0.2 ton;
```

12.3 Modifier Definitions

12.3.1 Modifier signed

```
32      modifier signed {
33          require(msg.pubkey() == tvm.pubkey(), 100);
34          tvm.accept();
35          _;
36      }
```

12.3.2 Modifier accept

- Minor issue: this modifier is dangerous in general, although not used in this project, because a function using it is easier to target to drain the balance of the contract. It should be removed.

```
38      modifier accept {
39          tvm.accept();
40          _;
41      }
```

12.3.3 Modifier onlyContract

```
43      modifier onlyContract() {
44          require(msg.sender != address(0), Errors.ONLY_CONTRACT);
45          _;
46      }
```

12.3.4 Modifier onlyMe

```
48      modifier onlyMe {
49          require(msg.sender == address(this), ERROR_DIFFERENT_CALLER
                );
50          _;
51      }
```

Chapter 13

Contract BftgRoot

Contents

13.1 Overview

In file `BftgRoot.sol`

13.2 Contract Inheritance

- Minor issue: `Checks` is not currently used. Remove it if there is no plan to use it.

Base	
IBftgRoot	
IBftgRootStoreCallback	
ContestResolver	
JuryGroupResolver	
Checks	

13.3 Constant Definitions

```
18    uint8 constant CHECK_CONTEST_CODE = 1;

19    uint8 constant CHECK_JURY_GROUP_CODE = 2;
```

13.4 Variable Definitions

```
34    address _addrBftgRootStore;

54    bool public _inited = false;

110   mapping(address => JuryGroupPending) _juryGroupPendings;
```

13.5 Modifier Definitions

13.5.1 Modifier onlyStore

```
29    modifier onlyStore() {
30        require(msg.sender == _addrBftgRootStore, Errors.ONLY_STORE
              );
31        _;
32    }
```

13.6 Constructor Definitions

13.6.1 Constructor

- **Critical issue: Administrative Take-over in `BftgRoot.constructor`**
  No test is performed to verify the sender in the case `msg.sender != address(0)`. An attacker could use it to deploy the contract himself for another user, providing its own `addrBftgRootStore`, i.e. with his own code for most contracts. Fix: contract should be deployed by the same public key as `tvm.pubkey` or the sender should be the same as a static variable `_deployer`.

- **Major issue: No initialization check performed in `BftgRoot.constructor`**
  The `_createChecks` function gives the false feeling the checks are performed for initialization of the Padawan and Proposal codes. However, the checks are not performed in the functions where they would be required. No attempt is done to perform the same checks for addresses.

```
36      constructor(address addrBftgRootStore) public {
37          if (msg.sender == address(0)) {
38              require(msg.pubkey() == tvm.pubkey(), Errors.
                    ONLY_SIGNED);
39          }
40          require(addrBftgRootStore != address(0), Errors.
                STORE_UNDEFINED);
41          tvm.accept();
42
43          _addrBftgRootStore = addrBftgRootStore;
44          IBftgRootStore(addrBftgRootStore).queryCode
45              {value: 0.2 ton, bounce: true}
46              (ContractCode.Contest);
47          IBftgRootStore(addrBftgRootStore).queryCode
48              {value: 0.2 ton, bounce: true}
49              (ContractCode.JuryGroup);
50
51          _createChecks();
52      }
```

13.7 Public Method Definitions

13.7.1 OnBounce function

- Minor issue: this function should check the message name being bounced.
- Minor issue (readability): _ should be avoided as a variable name.

```
83      onBounce(TvmSlice) external {
84          if(_juryGroupPendings.exists(msg.sender)) {
85              address[] _;
86              deployJuryGroup(_juryGroupPendings[msg.sender].tag, _);
87              this.registerMemberJuryGroup
88                  {value: 0, bounce: false, flag: 64}
89                  (_juryGroupPendings[msg.sender].tag,
                        _juryGroupPendings[msg.sender].addrJury);
90              delete _juryGroupPendings[msg.sender];
91          }
92      }
```

13.7.2 Function deployContest

- **Critical issue: tvm.accept without check in `BftgRoot.deployContest`**
  An attacker could drain the contract balance by sending many messages `deployContest`. Moreover, some of the arguments have unbounded size (`tags`), providing a way to make the attack even more efficient by sending large message with high gas cost. Fix: the sender should pay the gas.

```
98    function deployContest(string[] tags, uint128 prizePool, uint32
          underwayDuration) external view {
99        tvm.accept();
100       TvmCell state = _buildContestState(address(this));
101       new Contest
102           {stateInit: state, value: 1 ton}
103           (_addrBftgRootStore, tags, prizePool, underwayDuration)
                  ;
104   }
```

13.7.3 Function deployJuryGroup

- Minor issue: a `require` should check that there is enough value in the message to perform the deployment of the message.

```
112   function deployJuryGroup(string tag, address[] initialMembers)
          public view {
113       require(address(0) != msg.sender);
114       TvmCell state = _buildJuryGroupState(tag, address(this));
115       new JuryGroup
116           {stateInit: state, value: 0.3 ton}
117           (initialMembers);
118   }
```

13.7.4 Function getMembersCallback

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.
- Minor issue (gas cost): the argument `members` is not used in this function. It looks like asking for the list of members is only a way to check for the existence of the group. A less expensive function should be used instead of asking for the full list.

```
130   function getMembersCallback(mapping(address => Member) members)
          public {
131       require(_juryGroupPendings.exists(msg.sender) || address(
              this) == msg.sender, 106);
132       IJuryGroup(msg.sender).registerMember
133           {value: 0 ton, bounce: true, flag: 64}
134           (_juryGroupPendings[msg.sender].addrJury);
135       delete _juryGroupPendings[msg.sender];
136   }
```

13.7.5 Function getStored

- OK

```
142     function getStored() public view returns (
143         TvmCell codeContest,
144         TvmCell codeJuryGroup
145     ) {
146         codeContest = _codeContest;
147         codeJuryGroup = _codeJuryGroup;
148     }
```

13.7.6 Function registerMemberJuryGroup

- **Major issue: Non-reentrant in `BftgRoot.registerMemberJuryGroup`**
  If several `registerMemberJuryGroup` messages are sent together for the same `JuryGroup`, only the last one is taken into account, in `getMembersCallback`. This issue might lead to missing members, or to balance problems, given that multiple messages sent to `JuryGroup.registerMember` seems to be way to increase the balance for a particular member. Fix: either the contract should deal with multiple registration at the same time, or `registerMemberJuryGroup` should immediately fail if a registration is already in progress for the same group.
- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
120     function registerMemberJuryGroup(string tag, address addrMember
            ) public override {
121         address addrContest = resolveContest(address(this));
122         address addrJuryGroup = resolveJuryGroup(tag, address(this)
                );
123         require(msg.sender == addrContest || address(this) == msg.
                sender, 105);
124         _juryGroupPendings[addrJuryGroup] = JuryGroupPending(
                addrMember, tag);
125         IJuryGroup(addrJuryGroup).getMembers
126             {value: 0, bounce: true, flag: 64}
127             ();
128     }
```

13.7.7 Function updateAddr

- OK

```
77      function updateAddr(ContractAddr kind, address addr) external
            override {}
```

13.7.8 Function updateCode

- OK

```
62      function updateCode(
63          ContractCode kind,
64          TvmCell code
65      ) external override onlyStore {
66          if (kind == ContractCode.Contest) {
67              _codeContest = code;
68              _passCheck(CHECK_CONTEST_CODE);
69          }
70          if (kind == ContractCode.JuryGroup) {
71              _codeJuryGroup = code;
72              _passCheck(CHECK_JURY_GROUP_CODE);
73          }
74          _onInit();
75      }
```

13.8 Internal Method Definitions

13.8.1 Function _createChecks

- OK

```
21      function _createChecks() private inline {
22          _checkList = CHECK_CONTEST_CODE | CHECK_JURY_GROUP_CODE;
23      }
```

13.8.2 Function _onInit

- OK

```
56      function _onInit() private {
57          if(_isCheckListEmpty() && !_inited) {
58              _inited = true;
59          }
60      }
```

Chapter 14

Contract BftgRootStore

Contents

14.1 Overview

In file `BftgRootStore.sol`

14.2 General Minor-level Remarks

In general, the infrastructure would be safer if this contract would be implemented in two phases:

- In the Initialization phase, the contract is waiting for all the `setXXX` methods to be called to initialize all the fields. A bitmap can be used to keep the current initialization state. Any attempt to user a `getXXX` method should fail.
- In the Post-Initalization phase, the contract accepts to reply to `getXXX` methods, but `setXXX` methods are disabled.

There is also an inconsistency between the getters and setters: getters are generic (they take a `kind` as argument), whereas setters are specific (there is a different one for every kind).

14.3 Contract Inheritance

Base	
IBftgRootStore	

14.4 Variable Definitions

mapping (uint8 => address)	_addrs	
		used in @1.BftgRootStore.queryAddr
mapping (uint8 => TvmCell)	_codes	
		assigned in @1.BftgRootStore.setJuryGroupCode
		used in @1.BftgRootStore.setJuryGroupCode
		assigned in @1.BftgRootStore.setContestCode
		used in @1.BftgRootStore.setContestCode
		used in @1.BftgRootStore.queryCode

```
10      mapping(uint8 => address) public _addrs;

11      mapping(uint8 => TvmCell) public _codes;
```

14.5 Public Method Definitions

14.5.1 Function queryAddr

- Minor issue: a `require` could be added to fail if `kind` is not a well-known kind.

```
26      function queryAddr(ContractAddr kind) external override {
27          address addr = _addrs[uint8(kind)];
28          IBftgRootStoreCallback(msg.sender).updateAddr{value: 0,
                flag: 64, bounce: false}(kind, addr);
29      }
```

14.5.2 Function queryCode

- Minor issue: a `require` could be added to fail if `kind` is not a well-known kind.

```
21      function queryCode(ContractCode kind) external override {
22          TvmCell code = _codes[uint8(kind)];
23          IBftgRootStoreCallback(msg.sender).updateCode{value: 0,
                flag: 64, bounce: false}(kind, code);
24      }
```

14.5.3 Function setContestCode

- Minor issue: the infrastructure would probably be safer if the expected code hash is hardcoded in the source code, and check through a `require`

```
17      function setContestCode(TvmCell code) public override signed {
18          _codes[uint8(ContractCode.Contest)] = code;
19      }
```

14.5.4 Function setJuryGroupCode

- Minor issue: the infrastructure would probably be safer if the expected code hash is hardcoded in the source code, and check through a `require`

```
13      function setJuryGroupCode(TvmCell code) public override signed
            {
14          _codes[uint8(ContractCode.JuryGroup)] = code;
15      }
```

Chapter 15

Contract Checks

Contents

15.1 Overview

In file `Checks.sol`

This contract is now used directly, but only inherited by other contracts, such as `BftgRoot`. However, the checks are not used.

15.2 Variable Definitions

```
4     uint8 _checkList;
```

15.3 Modifier Definitions

15.3.1 Modifier checksEmpty

- Minor issue: a `tvm.accept` should not be used without checking the origin of the message. Here, the checks are only done on the current initialization of the contract. In general, such a modifier could be used by an attacker to drain the balance of the contract. We advise to either remove the modifier, or remove the call to `tvm.accept`.

```
12      modifier checksEmpty() {
13          require(_isCheckListEmpty(), 100); //Errors.
                NOT_ALL_CHECKS_PASSED);
14          tvm.accept();
15          _;
16      }
```

15.4 Internal Method Definitions

15.4.1 Function _isCheckListEmpty

- OK

```
9       function _isCheckListEmpty() internal view inline returns (bool
            ) {
10          return (_checkList == 0);
11      }
```

15.4.2 Function _passCheck

- OK

```
6       function _passCheck(uint8 check) internal inline {
7           _checkList &= ~check;
8       }
```

Chapter 16

Contract Contest

Contents

16.1 Overview

In file `Contest.sol`

16.2 Contract Inheritance

- Minor issue: `Checks` is not currently used. Remove it if there is no plan to use it.

JuryGroupResolver	
IJuryGroupCallback	
IBftgRootStoreCallback	
Checks	

16.3 Constant Definitions

- OK

```
15    uint8 constant CHECK_JURY_GROUP_CODE = 1;
```

16.4 Static Variable Definitions

- OK

```
28    address static _deployer;
```

16.5 Variable Definitions

- OK

```
25    string[] public _tags;
26    mapping(address => bool) _tagsPendings;
27    mapping(address => Member) public _jury;
30    uint128 public _prizePool;
31    uint32 public _underwayDuration;
32    uint32 public _underwayEnds;
45    bool public _inited = false;
```

```
104     ContestStage public _stage;

118     mapping(uint32 => Submission) public _submissions;

119     uint32 _submissionsCounter;

132     mapping(address => mapping(uint32 => HiddenVote)) public
            _juryHiddenVotes;

153     mapping(uint32 => Vote[]) public _submissionVotes;

171     uint128 _pointValue;

173     mapping(address => Reward) public _rewards;
```

16.6 Constructor Definitions

16.6.1 Constructor

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
34      constructor(address addrBftgRootStore, string[] tags, uint128
            prizePool, uint32 underwayDuration) public {
35          require(msg.sender == _deployer, 101);
36          _tags = tags;
37          _stage = ContestStage.New;
38          _prizePool = prizePool;
39          _underwayDuration = underwayDuration;
40          IBftgRootStore(addrBftgRootStore).queryCode
41              {value: 0.2 ton, bounce: true}
42              (ContractCode.JuryGroup);
43      }
```

16.7 Public Method Definitions

16.7.1 OnBounce function

- Minor issue: this function should check the message name being bounced.

```
64      onBounce(TvmSlice) external {
65          if(_tagsPendings.exists(msg.sender)) {
66              delete _tagsPendings[msg.sender];
67              if(_tagsPendings.empty()) {
68                  _changeStage(ContestStage.Underway);
69              }
70          }
71      }
```

16.7.2 Function calcRewards

- **Critical issue: No stage check in `Contest.calcRewards`**
  Because this function performs no check on the sender, and no check on the current stage (except the one of monotonicity in `_changeStage`), an attacker could use it to terminate a contest from any stage before the `Reward` stage to that stage without passing through previous stages. Fix: this function should check for a delay after the start of the voting stage.

- **Major issue: Wrong computation in `Contest.calcRewards`**
  The interpretation of "point value" differs in `calcRewards` and `_calcPointValue`. Indeed, in `_calcPointValue`, the "point value" is the value of a point for the **average** submission score, whereas `calcRewards` uses it for every point of a submission vote, i.e. not the average. Though the computation in `_calcPointValue` is not the final one, this difference in interpretation may lead to rewards much higher than the ones expected.

```
175     function calcRewards() public {
176         _calcPointValue();
177         optional(uint32, Vote[]) optSubmissionVotes =
                _submissionVotes.min();
178         while (optSubmissionVotes.hasValue()) {
179             (uint32 id, Vote[] submissionVotes) =
                    optSubmissionVotes.get();
180             for(uint8 i = 0; i < submissionVotes.length; i++) {
181                 _rewards[_submissions[id].addrPartisipant].total +=
                        submissionVotes[i].score * _pointValue;
182             }
183             optSubmissionVotes = _submissionVotes.next(id);
184         }
185         _changeStage(ContestStage.Reward);
186     }
```

16.7.3 Function changeStage

- **Critical issue: Missing permission checks in `Contest.changeStage`**
  No permission checks are performed in this function. An attacker could freely change the stage of the contest, and drain the message balance using `tvm.accept`.

```
234     function changeStage(ContestStage stage) external {
235         tvm.accept();
236         _stage = stage;
237     }
```

16.7.4 Function claimPartisipantReward

- Minor issue: fix spelling of `participant` instead of `partisipant`.

```
197     function claimPartisipantReward(uint128 amount) public {
198         require(_rewards.exists(msg.sender), 107);
199         require(_rewards[msg.sender].total - _rewards[msg.sender].
                paid >= amount, 108);
200         _rewards[msg.sender].paid += amount;
201         msg.sender.transfer(amount, true, 1);
202     }
```

16.7.5 Function getHiddenVotesByAddress

- OK

```
145     function getHiddenVotesByAddress(address juryAddr) public view
            returns (mapping(uint32 => HiddenVote) hiddenVotes) {
146         hiddenVotes = _juryHiddenVotes[juryAddr];
147     }
```

16.7.6 Function getMembersCallback

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.
- Minor issue: the test `member.balance >= 0` is useless as the field is an unsigned integer `uint128`.

```
87      function getMembersCallback(mapping(address => Member) members)
             external override {
88          require(_tagsPendings.exists(msg.sender), 102);
89          delete _tagsPendings[msg.sender];
90          for((, Member member): members) {
91              if(member.balance >= 0) {
92                  _jury[member.addr] = member;
93              }
94          }
95          if(_tagsPendings.empty()) {
96              _changeStage(ContestStage.Underway);
97          }
98      }
```

16.7.7 Function hashVote

- OK

```
223     function hashVote(uint32 submissionId, uint8 score, string
            comment) public pure returns (uint hash) {
224         TvmBuilder builder;
225         builder.store(submissionId, score, comment);
226         TvmCell cell = builder.toCell();
227         hash = tvm.hash(cell);
228     }
```

16.7.8 Function reveal

- **Critical issue: Multiple revelations in `Contest.reveal`**
  A jury can reveal his votes several times, adding them several times in the `_submissionVotes` table. Fix: remove submission from `_juryHiddenVotes` everytime they are revealed.
- Minor issue (gas cost): instead of failing if `oldHash` and `newHash` differ, the function should probably returns the list of failed couples, and keep working for correct couples.
- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
155     function reveal(RevealVote[] revealVotes) external {
156         require(_stage == ContestStage.Reveal, 104);
157         require(_jury.exists(msg.sender), 105);
158         for(uint8 i = 0; i < revealVotes.length; i++) {
159             uint oldHash = _juryHiddenVotes[msg.sender][revealVotes
                    [i].submissionId].hash;
160             uint newHash = hashVote(revealVotes[i].submissionId,
                    revealVotes[i].score, revealVotes[i].comment);
161             require(oldHash == newHash, 106);
162             _submissionVotes[revealVotes[i].submissionId].push(Vote
                    (msg.sender, revealVotes[i].score, revealVotes[i].
                    comment));
163         }
164         msg.sender.transfer(0, true, 64);
165     }
```

16.7.9 Function stakePartisipantReward

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
204     function stakePartisipantReward(uint128 amount, string tag,
            address addrJury) public {
205         require(_rewards.exists(msg.sender), 107);
206         require(_rewards[msg.sender].total - _rewards[msg.sender].
                paid >= amount, 108);
207         bool isTagExists = false;
208         for(uint8 i = 0; i < _tags.length; i++) {
209             if(_tags[i] == tag) isTagExists = true;
210         }
211         require(isTagExists, 108);
212         _rewards[msg.sender].paid += amount;
213         IBftgRoot(_deployer).registerMemberJuryGroup
214             {value: amount, bounce: true, flag: 2}
215             (tag, addrJury == address(0) ? msg.sender : addrJury);
216         msg.sender.transfer(0, true, 64);
217     }
```

16.7.10 Function submit

- **Major issue: Unbounded storage in `Contest.submit`**
  Anybody can call this function. An attacker could use it to increase dramatically the cost of calling the contract by storing a very big submission into the contest storage.
- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
121     function submit(address addrPartisipant, string forumLink,
            string fileLink, uint hash) external {
122         require(_stage == ContestStage.Underway, 104);
123         _submissions[_submissionsCounter] = (Submission(
                _submissionsCounter, addrPartisipant, forumLink,
                fileLink, hash, uint32(now)));
124         _submissionsCounter += 1;
125         msg.sender.transfer(0, true, 64);
126     }
```

16.7.11 Function updateAddr

- OK

```
81      function updateAddr(ContractAddr kind, address addr) external
            override {}
```

16.7.12 Function updateCode

- **Critical issue: No permission check in `Contest.updateCode`**
  No check is performed on the sender of this message, allowing an attacker to provide his own malicious implementation of `JuryGroup` to the contract. Fix: check the sender, or check the code hash of the code.
- **Major issue: No gas check in `Contest.updateCode`**
  Given that this function is responsible for sending `getMembers` messages to all jury groups, it should check by `require` that the message contains enough gas to perform these sends. Otherwise, it could happen that the action phase could succeed, the contract would remember that it was initialized, yet the transaction would be aborted in the sending phase and no message would actually be sent by lack of gas.
- Minor issue: the infrastructure would probably be safer if the expected code hash is hardcoded in the source code, and check through a `require`
- Minor issue: if `kind` is not `ContractCode.JuryGroup`, this function will silently return without error, whereas the user might interpret it as successful and initialization done. Fix: replace the `if` by a `require`.

```
73      function updateCode(ContractCode kind, TvmCell code) external
            override {
74          if (kind == ContractCode.JuryGroup) {
75              _codeJuryGroup = code;
76              _passCheck(CHECK_JURY_GROUP_CODE);
77          }
78          _onInit();
79      }
```

16.7.13 Function vote

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.
- Minor issue: maybe this function could be relaxed to allow the voter to change his vote

```
134     function vote(HiddenVote[] hiddenVotes) external {
135         require(_stage == ContestStage.Voting, 104);
136         require(_jury.exists(msg.sender), 105);
137         for(uint8 i = 0; i < hiddenVotes.length; i++) {
138             if(!_juryHiddenVotes[msg.sender].exists(hiddenVotes[i].
                    submissionId)) {
139                 _juryHiddenVotes[msg.sender][hiddenVotes[i].
                        submissionId] = hiddenVotes[i];
140             }
141         }
142         msg.sender.transfer(0, true, 64);
143     }
```

16.8 Internal Method Definitions

16.8.1 Function _calcPointValue

- OK

```
188     function _calcPointValue() private inline {
189         // TODO: change the formula
190         _pointValue = _prizePool / (_submissionsCounter * 10);
191     }
```

16.8.2 Function _changeStage

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
106     function _changeStage(ContestStage stage) private inline
            returns (ContestStage) {
107         require(_stage < stage, 103);
108         if (stage == ContestStage.Underway) {
109             _underwayEnds = uint32(now) + _underwayDuration;
110         }
111         _stage = stage;
112     }
```

16.8.3 Function _createChecks

- OK

```
17     function _createChecks() private inline {
18         _checkList = CHECK_JURY_GROUP_CODE;
19     }
```

16.8.4 Function _onInit

- OK

```
47     function _onInit() private {
48         if(_isCheckListEmpty() && !_inited) {
49             _inited = true;
50             for(uint8 i = 0; i < _tags.length; i++) {
51                 TvmCell state = _buildJuryGroupState(_tags[i],
                       _deployer);
52                 uint256 hashState = tvm.hash(state);
53                 address addrJuryGroup = address.makeAddrStd(0,
                       hashState);
54                 _tagsPendings[addrJuryGroup] = true;
55                 IJuryGroup(addrJuryGroup).getMembers{
56                     value: 0.2 ton,
57                     flag: 1,
58                     bounce: true
59                 }();
60             }
61         }
62     }
```

Chapter 17

Contract ContestResolver

Contents

17.1 Overview

In file `ContestResolver.sol`

17.2 Variable Definitions

```
8     TvmCell _codeContest;
```

17.3 Public Method Definitions

17.3.1 Function resolveContest

- OK

```
10    function resolveContest(address deployer) public view returns (
          address addrContest) {
11        TvmCell state = _buildContestState(deployer);
12        uint256 hashState = tvm.hash(state);
13        addrContest = address.makeAddrStd(0, hashState);
14    }
```

17.4 Internal Method Definitions

17.4.1 Function _buildContestState

- Minor issue: this function should fail (`require`) if the `_codeContest` variable has not yet been initialized. A global boolean could be used for that, set in an internal function initializing both global variables.

```
16      function _buildContestState(address deployer) internal virtual
            view returns (TvmCell) {
17          return tvm.buildStateInit({
18              contr: Contest,
19              varInit: {_deployer: deployer},
20              code: _codeContest
21          });
22      }
```

Chapter 18

Contract JuryGroup

Contents

18.1 Overview

In file `JuryGroup.sol`

18.2 Contract Inheritance

IJuryGroup	

18.3 Static Variable Definitions

```
11      string static public _tag;

12      address static _deployer;
```

18.4 Variable Definitions

```
14      mapping(address => Member) public _members;

15      uint32 _membersCounter;
```

18.5 Modifier Definitions

18.5.1 Modifier onlyDeployer

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
6       modifier onlyDeployer() {
7           require(msg.sender == _deployer, 100);
8           _;
9       }
```

18.6 Constructor Definitions

18.6.1 Constructor

- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

```
17      constructor(address[] initialMembers) public {
18          require(_deployer == msg.sender, 100);
19          for(uint8 i = 0; i < initialMembers.length; i++) {
20              _addMember(initialMembers[i], 0);
21          }
22      }
```

18.7 Public Method Definitions

18.7.1 Function getMembers

- OK

```
45      function getMembers() public override {
46          IJuryGroupCallback(msg.sender).getMembersCallback{value: 0,
                flag: 64, bounce: false}(_members);
47      }
```

18.7.2 Function registerMember

- Minor issue (readability): replace the comparison with `false` by inversing the `then` and `else` clauses in the `if`

```
24      function registerMember(address addrMember) public override
            onlyDeployer {
25          if(_members.exists(addrMember) == false) {
26              _addMember(addrMember, msg.value);
27          } else {
28              _members[addrMember].balance += msg.value;
29          }
30      }
```

18.7.3 Function withdraw

- **Major issue: Wrong comparison in `JuryGroup.withdraw`**
  The check `_members[msg.sender].balance < amount` will fail, or if it does not fail, the operation `_members[msg.sender].balance -= amount` will fail. Either way, the function will always fail.
- Minor issue: the check `_members[msg.sender].balance >= 0 ton` is always true, because `balance` is an `uint128`.

```
37      function withdraw(uint128 amount) public {
38          require(msg.sender != address(0), 101);
39          require(_members[msg.sender].balance >= 0 ton, 201);
40          require(_members[msg.sender].balance < amount, 202);
41          msg.sender.transfer(amount, true, 1);
42          _members[msg.sender].balance -= amount;
43      }
```

18.8 Internal Method Definitions

18.8.1 Function _addMember

- OK

```
32      function _addMember(address addrMember, uint128 value) private
            inline {
33          _members[addrMember] = Member(_membersCounter, value,
                addrMember);
34          _membersCounter++;
35      }
```

Chapter 19

Contract JuryGroupResolver

Contents

19.1 Overview

In file `JuryGroupResolver.sol`

19.2 Variable Definitions

```
6     TvmCell _codeJuryGroup;
```

19.3 Public Method Definitions

19.3.1 Function resolveJuryGroup

- OK

```
8     function resolveJuryGroup(string tag, address deployer) public
          view returns (address addrJuryGroup) {
9         TvmCell state = _buildJuryGroupState(tag, deployer);
```

```
10          uint256 hashState = tvm.hash(state);
11          addrJuryGroup = address.makeAddrStd(0, hashState);
12      }
```

19.4 Internal Method Definitions

19.4.1 Function _buildJuryGroupState

- Minor issue: this function should fail (`require`) if the `_codeJuryGroup` variable has not yet been initialized. A global boolean could be used for that, set in an internal function initializing both global variables.

```
14      function _buildJuryGroupState(string tag, address deployer)
            internal view returns (TvmCell) {
15          return tvm.buildStateInit({
16              contr: JuryGroup,
17              varInit: {_tag: tag, _deployer: deployer},
18              code: _codeJuryGroup
19          });
20      }
```