Audit of the BFTG project

By OCamlPro

August 20, 2021

Table of Major and Critical Issues

Major issue: Incorrect computation in `Padawan.onEstimateVotes` 8
Critical issue: Can empty voting rights in `Padawan.onTokenWalletDeploy` 9
Critical issue: Unbounded voting rights in `Padawan.onTokenWalletGetBalance` 9
Critical issue: Race condition in `Padawan.reclaimDeposit` . . . 10

Critical issue: No permission check on `Proposal.onGetMembers` 14
Critical issue: Division by 0 in `Proposal._softMajority` 16

Critical issue: No permission check in `Group.constructor` . . . 18
Critical issue: No permission check in `Group.addMember` 19
Critical issue: No permission check on `removeMember` 19

Critical issue: Administrative Take-over in `SmvRoot.constructor` 20
Major issue: No check on `SmvRoot.deployGroup` 22

Critical issue: Administrative Take-over in `BftgRoot.constructor` 35
Major issue: No initialization check performed in `BftgRoot.constructor` 35
Critical issue: tvm.accept without check in `BftgRoot.deployContest` 37
Major issue: Non-reentrant in `BftgRoot.registerMemberJuryGroup` 38

Critical issue: No stage check in `Contest.calcRewards` 43
Major issue: Wrong computation in `Contest.calcRewards` . . . 43
Critical issue: Missing permission checks in `Contest.changeStage` 44
Critical issue: Multiple revelations in `Contest.reveal` 45
Major issue: Unbounded storage in `Contest.submit` 46
Critical issue: No permission check in `Contest.updateCode` . . 46
Major issue: No gas check in `Contest.updateCode` 46

Major issue: Wrong comparison in `JuryGroup.withdraw` 50

Contents

1 Overview **6**
- 1.1 Source code location . . . 6
- 1.2 Architecture . . . 6

2 Contract Padawan **7**
- 2.1 Overview . . . 7
- 2.2 Public Method Definitions . . . 7
  - 2.2.1 Function confirmVote . . . 7
  - 2.2.2 Function onEstimateVotes . . . 8
  - 2.2.3 Function onTokenWalletDeploy . . . 9
  - 2.2.4 Function onTokenWalletGetBalance . . . 9
  - 2.2.5 Function reclaimDeposit . . . 10
  - 2.2.6 Function rejectVote . . . 11
  - 2.2.7 Function updateStatus . . . 11

3 Contract Proposal **13**
- 3.1 Overview . . . 13
- 3.2 Constructor Definitions . . . 13
  - 3.2.1 Constructor . . . 13
- 3.3 Public Method Definitions . . . 14
  - 3.3.1 Function onGetMembers . . . 14
  - 3.3.2 Function queryStatus . . . 15
  - 3.3.3 Function vote . . . 15
- 3.4 Internal Method Definitions . . . 16
  - 3.4.1 Function _softMajority . . . 16
  - 3.4.2 Function _tryEarlyComplete . . . 16
  - 3.4.3 Function _wrapUp . . . 17

4 Contract Group **18**
- 4.1 Overview . . . 18
- 4.2 Constructor Definitions . . . 18
  - 4.2.1 Constructor . . . 18
- 4.3 Public Method Definitions . . . 19
  - 4.3.1 Function addMember . . . 19

4.3.2 Function removeMember 19

5 Contract SmvRoot **20**
5.1 Overview 20
5.2 Constructor Definitions 20
5.2.1 Constructor 20
5.3 Public Method Definitions 21
5.3.1 Function _deployProposal 21
5.3.2 Function deployGroup 22
5.3.3 Function deployPadawan 22
5.3.4 Function deployProposal 22
5.4 Internal Method Definitions 23
5.4.1 Function _beforeProposalDeploy 23

6 Contract PadawanResolver **25**
6.1 Overview 25
6.2 Internal Method Definitions 25
6.2.1 Function _buildPadawanState 25

7 Contract ProposalFactoryResolver **26**
7.1 Overview 26
7.2 Internal Method Definitions 26
7.2.1 Function _buildProposalFactoryState 26

8 Contract ProposalResolver **27**
8.1 Overview 27
8.2 Internal Method Definitions 27
8.2.1 Function _buildProposalState 27

9 Contract GroupResolver **28**
9.1 Overview 28
9.2 Internal Method Definitions 28
9.2.1 Function _buildGroupState 28

10 Contract SmvRootStore **29**
10.1 Overview 29
10.2 General Minor-level Remarks 29
10.3 Public Method Definitions 30
10.3.1 Function queryAddr 30
10.3.2 Function queryCode 30
10.3.3 Function setGroupCode 30
10.3.4 Function setPadawanCode 30
10.3.5 Function setProposalCode 31
10.3.6 Function setProposalFactoryCode 31

11 Contract Base **32**
11.1 Overview . . . 32
11.2 Constant Definitions . . . 32
11.3 Modifier Definitions . . . 33
11.3.1 Modifier signed . . . 33
11.3.2 Modifier accept . . . 33
11.3.3 Modifier onlyContract . . . 34
11.3.4 Modifier onlyMe . . . 34

12 Contract BftgRoot **35**
12.1 Overview . . . 35
12.2 Constructor Definitions . . . 35
12.2.1 Constructor . . . 35
12.3 Public Method Definitions . . . 36
12.3.1 OnBounce function . . . 36
12.3.2 Function deployContest . . . 37
12.3.3 Function deployJuryGroup . . . 37
12.3.4 Function getMembersCallback . . . 37
12.3.5 Function registerMemberJuryGroup . . . 38

13 Contract BftgRootStore **39**
13.1 Overview . . . 39
13.2 General Minor-level Remarks . . . 39
13.3 Public Method Definitions . . . 40
13.3.1 Function queryAddr . . . 40
13.3.2 Function queryCode . . . 40
13.3.3 Function setContestCode . . . 40
13.3.4 Function setJuryGroupCode . . . 40

14 Contract Checks **41**
14.1 Overview . . . 41
14.2 Modifier Definitions . . . 41
14.2.1 Modifier checksEmpty . . . 41

15 Contract Contest **42**
15.1 Overview . . . 42
15.2 Constructor Definitions . . . 42
15.2.1 Constructor . . . 42
15.3 Public Method Definitions . . . 43
15.3.1 OnBounce function . . . 43
15.3.2 Function calcRewards . . . 43
15.3.3 Function changeStage . . . 44
15.3.4 Function claimPartisipantReward . . . 44
15.3.5 Function getMembersCallback . . . 44
15.3.6 Function reveal . . . 45
15.3.7 Function stakePartisipantReward . . . 45

15.3.8 Function submit 46
15.3.9 Function updateCode 46
15.3.10 Function vote 47
15.4 Internal Method Definitions 47
15.4.1 Function _changeStage 47

16 Contract ContestResolver **48**
16.1 Overview 48
16.2 Internal Method Definitions 48
16.2.1 Function _buildContestState 48

17 Contract JuryGroup **49**
17.1 Overview 49
17.2 Modifier Definitions 49
17.2.1 Modifier onlyDeployer 49
17.3 Constructor Definitions 50
17.3.1 Constructor 50
17.4 Public Method Definitions 50
17.4.1 Function registerMember 50
17.4.2 Function withdraw 50

18 Contract JuryGroupResolver **51**
18.1 Overview 51
18.2 Internal Method Definitions 51
18.2.1 Function _buildJuryGroupState 51

Chapter 1

Overview

1.1 Source code location

The source code is available `https://github.com/RSquad/BFTG` at branch master with hash code equal to 7c6ec7d811bcc1f228a3499ab19f6d20652ca94b

1.2 Architecture

The architecture is mostly similar to the one detailed in the DENS-SMV audit report, with only a few differences:

- The `Demiurge` contract of DENS-SMV is called `SmvRoot` contract
- Voting can be restricted to a white list specified either directly or through a `Group` contract
- A `Padawan` contract can manage multiple token wallets, so that the user may have different voting rights for different proposals

Chapter 2

Contract Padawan

Contents

2.1 Overview 7
2.2 Public Method Definitions 7
2.2.1 Function confirmVote 7
2.2.2 Function onEstimateVotes 8
2.2.3 Function onTokenWalletDeploy 9
2.2.4 Function onTokenWalletGetBalance 9
2.2.5 Function reclaimDeposit 10
2.2.6 Function rejectVote 11
2.2.7 Function updateStatus 11

2.1 Overview

In file `Padawan.sol`

2.2 Public Method Definitions

2.2.1 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     }
```

2.2.2 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      }
```

2.2.3 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     }
```

2.2.4 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     }
```

2.2.5 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     }
```

2.2.6 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     }
```

2.2.7 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     }
```

Chapter 3

Contract Proposal

Contents

3.1 Overview 13
3.2 Constructor Definitions 13
3.2.1 Constructor 13
3.3 Public Method Definitions 14
3.3.1 Function onGetMembers 14
3.3.2 Function queryStatus 15
3.3.3 Function vote 15
3.4 Internal Method Definitions 16
3.4.1 Function _softMajority 16
3.4.2 Function _tryEarlyComplete 16
3.4.3 Function _wrapUp 17

3.1 Overview

In file `Proposal.sol`

3.2 Constructor Definitions

3.2.1 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      }
```

3.3 Public Method Definitions

3.3.1 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     }
```

3.3.2 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     }
```

3.3.3 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     }
```

3.4 Internal Method Definitions

3.4.1 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     }
```

3.4.2 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     }
```

3.4.3 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 18
4.2 Constructor Definitions 18
4.2.1 Constructor 18
4.3 Public Method Definitions 19
4.3.1 Function addMember 19
4.3.2 Function removeMember 19

4.1 Overview

In file `Group.sol`

4.2 Constructor Definitions

4.2.1 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      }
```

4.3 Public Method Definitions

4.3.1 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      }
```

4.3.2 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 20
5.2 Constructor Definitions 20
5.2.1 Constructor 20
5.3 Public Method Definitions 21
5.3.1 Function _deployProposal 21
5.3.2 Function deployGroup 22
5.3.3 Function deployPadawan 22
5.3.4 Function deployProposal 22
5.4 Internal Method Definitions 23
5.4.1 Function _beforeProposalDeploy 23

5.1 Overview

In file `SmvRoot.sol`

5.2 Constructor Definitions

5.2.1 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      }
```

5.3 Public Method Definitions

5.3.1 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     }
```

5.3.2 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     }
```

5.3.3 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     }
```

5.3.4 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     }
```

5.4 Internal Method Definitions

5.4.1 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     }
```

Chapter 6

Contract PadawanResolver

Contents

6.1 Overview 25
6.2 Internal Method Definitions 25
6.2.1 Function _buildPadawanState 25

6.1 Overview

In file `PadawanResolver.sol`

6.2 Internal Method Definitions

6.2.1 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 7

Contract ProposalFactoryResolver

Contents

7.1 Overview 26
7.2 Internal Method Definitions 26
7.2.1 Function _buildProposalFactoryState 26

7.1 Overview

In file `ProposalFactoryResolver.sol`

7.2 Internal Method Definitions

7.2.1 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 8

Contract ProposalResolver

Contents

8.1 Overview 27
8.2 Internal Method Definitions 27
8.2.1 Function _buildProposalState 27

8.1 Overview

In file `ProposalResolver.sol`

8.2 Internal Method Definitions

8.2.1 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 9

Contract GroupResolver

Contents

9.1 Overview 28
9.2 Internal Method Definitions 28
9.2.1 Function _buildGroupState 28

9.1 Overview

In file `GroupResolver.sol`

9.2 Internal Method Definitions

9.2.1 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 10

Contract SmvRootStore

Contents

10.1 Overview 29
10.2 General Minor-level Remarks 29
10.3 Public Method Definitions 30
10.3.1 Function queryAddr 30
10.3.2 Function queryCode 30
10.3.3 Function setGroupCode 30
10.3.4 Function setPadawanCode 30
10.3.5 Function setProposalCode 31
10.3.6 Function setProposalFactoryCode 31

10.1 Overview

In file `SmvRootStore.sol`

10.2 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).

10.3 Public Method Definitions

10.3.1 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      }
```

10.3.2 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      }
```

10.3.3 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      }
```

10.3.4 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      }
```

10.3.5 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      }
```

10.3.6 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 11

Contract Base

Contents

11.1 Overview 32
11.2 Constant Definitions 32
11.3 Modifier Definitions 33
11.3.1 Modifier signed 33
11.3.2 Modifier accept 33
11.3.3 Modifier onlyContract 34
11.3.4 Modifier onlyMe 34

11.1 Overview

In file `Base.sol`

11.2 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;
```

11.3 Modifier Definitions

11.3.1 Modifier signed

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

11.3.2 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      }
```

11.3.3 Modifier onlyContract

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

11.3.4 Modifier onlyMe

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

Chapter 12

Contract BftgRoot

Contents

12.1 Overview 35
12.2 Constructor Definitions 35
12.2.1 Constructor 35
12.3 Public Method Definitions 36
12.3.1 OnBounce function 36
12.3.2 Function deployContest 37
12.3.3 Function deployJuryGroup 37
12.3.4 Function getMembersCallback 37
12.3.5 Function registerMemberJuryGroup 38

12.1 Overview

In file `BftgRoot.sol`

12.2 Constructor Definitions

12.2.1 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      }
```

12.3 Public Method Definitions

12.3.1 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      }
```

12.3.2 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   }
```

12.3.3 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   }
```

12.3.4 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   }
```

12.3.5 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     }
```

Chapter 13

Contract BftgRootStore

Contents

13.1 Overview 39
13.2 General Minor-level Remarks 39
13.3 Public Method Definitions 40
13.3.1 Function queryAddr 40
13.3.2 Function queryCode 40
13.3.3 Function setContestCode 40
13.3.4 Function setJuryGroupCode 40

13.1 Overview

In file `BftgRootStore.sol`

13.2 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).

13.3 Public Method Definitions

13.3.1 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      }
```

13.3.2 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      }
```

13.3.3 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      }
```

13.3.4 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 14

Contract Checks

Contents

14.1 Overview 41
14.2 Modifier Definitions 41
14.2.1 Modifier checksEmpty 41

14.1 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.

14.2 Modifier Definitions

14.2.1 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      }
```

Chapter 15

Contract Contest

Contents

15.1 Overview 42
15.2 Constructor Definitions 42
15.2.1 Constructor 42
15.3 Public Method Definitions 43
15.3.1 OnBounce function 43
15.3.2 Function calcRewards 43
15.3.3 Function changeStage 44
15.3.4 Function claimPartisipantReward 44
15.3.5 Function getMembersCallback 44
15.3.6 Function reveal 45
15.3.7 Function stakePartisipantReward 45
15.3.8 Function submit 46
15.3.9 Function updateCode 46
15.3.10 Function vote 47
15.4 Internal Method Definitions 47
15.4.1 Function _changeStage 47

15.1 Overview

In file `Contest.sol`

15.2 Constructor Definitions

15.2.1 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      }
```

15.3 Public Method Definitions

15.3.1 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      }
```

15.3.2 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     }
```

15.3.3 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     }
```

15.3.4 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     }
```

15.3.5 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      }
```

15.3.6 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     }
```

15.3.7 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     }
```

15.3.8 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     }
```

15.3.9 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      }
```

15.3.10 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     }
```

15.4 Internal Method Definitions

15.4.1 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     }
```

Chapter 16

Contract ContestResolver

Contents

16.1 Overview 48
16.2 Internal Method Definitions 48
16.2.1 Function _buildContestState 48

16.1 Overview

In file `ContestResolver.sol`

16.2 Internal Method Definitions

16.2.1 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 17

Contract JuryGroup

Contents

17.1 Overview 49
17.2 Modifier Definitions 49
17.2.1 Modifier onlyDeployer 49
17.3 Constructor Definitions 50
17.3.1 Constructor 50
17.4 Public Method Definitions 50
17.4.1 Function registerMember 50
17.4.2 Function withdraw 50

17.1 Overview

In file `JuryGroup.sol`

17.2 Modifier Definitions

17.2.1 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     }
```

17.3 Constructor Definitions

17.3.1 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      }
```

17.4 Public Method Definitions

17.4.1 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      }
```

17.4.2 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      }
```

Chapter 18

Contract JuryGroupResolver

Contents

18.1 Overview 51
18.2 Internal Method Definitions 51
18.2.1 Function _buildJuryGroupState 51

18.1 Overview

In file `JuryGroupResolver.sol`

18.2 Internal Method Definitions

18.2.1 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      }
```