Audit

By OCamlPro

August 19, 2021

Table of Major and Critical Issues

Contents

To edit this document

In the report.tex file, choose:

- **\soldraftfalse** to remove draft mode (watermarks, advises)
- **\solmodulestrue** to display modules by chapter instead of contracts
- **\soltablestrue** to display tables for parameters and returns
- **\solissuesfalse** to remove the table of issues

Issues can be entered with:

- **\issueCritical{title}{text}**
- **\issueMajor{title}{text}**
- **\issueMinor{title}{text}**

Chapter 1

Introduction

1.0.1 Location

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

1.0.2 End Date

The contest ends at Aug 20, 2021, 23:59:59 UTC

Chapter 2

Overview

Chapter 3

Library Modules

3.1 Module "BFTG.sol"

3.1.1 Imports

../../BFTG/src/BftgRoot.sol	
../../BFTG/src/Padawan.sol	
../../BFTG/src/Proposal.sol	

3.2 Module "Errors.sol"

3.2.1 Pragmas

ton	-solidity >=0.37.0	

3.2.2 Contract Definitions

- Errors

3.3 Module "Glossary.sol"

3.3.1 Pragmas

ton	-solidity >= 0.36.0	

3.3.2 Type Definitions

3.3.2.1 Enum VoteCountModel

Undefined	
Majority	
SoftMajority	
SuperMajority	
Other	
Reserved	
Last	

```
3  enum VoteCountModel {
4      Undefined,
5      Majority,
6      SoftMajority,
7      SuperMajority,
8      Other,
9      Reserved,
10     Last
11 }
```

3.3.2.2 Enum ProposalType

Undefined	
SetCode	
Reserve	
SetOwner	
SetRootOwner	

```
13 enum ProposalType {
14     Undefined,
15     SetCode,
16     Reserve,
17     SetOwner,
18     SetRootOwner
19 }
```

3.3.2.3 Enum ProposalState

Undefined	
New	
OnVoting	
Ended	
Passed	
NotPassed	
Finalized	
Distributed	
Reserved	
Last	

```
21 enum ProposalState {
22     Undefined,
23     New,
24     OnVoting,
25     Ended,
26     Passed,
27     NotPassed,
28     Finalized,
29     Distributed,
30     Reserved,
31     Last
32 }
```

3.4 Module "IContest.sol"

3.4.1 Pragmas

ton	-solidity >= 0.42.0	

3.4.2 Type Definitions

3.4.2.1 Enum ContestStage

Undefined	
New	
Underway	
Voting	
Reveal	
Rank	
Reward	
Finish	
Last	

```
3  enum ContestStage {
4      Undefined,
5      New,
6      Underway,
7      Voting,
8      Reveal,
9      Rank,
10     Reward,
11     Finish,
12     Last
13 }
```

3.4.2.2 Struct Submission

id	uint32	
addrPartisipant	address	
forumLink	string	
fileLink	string	
hash	uint256	
createdAt	uint32	

```
15 struct Submission {
16   uint32 id;
17   address addrPartisipant;
18   string forumLink;
19   string fileLink;
20   uint hash;
21   uint32 createdAt;
22 }
```

3.4.2.3 Struct HiddenVote

submissionId	uint32	
hash	uint256	
hiddenComment	bytes	
hiddenScore	bytes	

```
24 struct HiddenVote {
25     uint32 submissionId;
26     uint hash;
27     bytes hiddenComment;
28     bytes hiddenScore;
29 }
```

3.4.2.4 Struct RevealVote

submissionId	uint32	
score	uint8	
comment	bytes	

```
31 struct RevealVote {
32     uint32 submissionId;
33     uint8 score;
34     bytes comment;
35 }
```

3.4.2.5 Struct Vote

addrJury	address	
score	uint8	
comment	bytes	

```
37 struct Vote {
38     address addrJury;
39     uint8 score;
40     bytes comment;
41 }
```

3.4.2.6 Struct Reward

total	uint128	
paid	uint128	

```
43 struct Reward {
44     uint128 total;
45     uint128 paid;
46 }
```

Chapter 4

Interface Modules

4.1 Module "IBftgRoot.sol"

4.1.1 Pragmas

ton	-solidity >= 0.42.0	

4.1.2 Type Definitions

4.1.2.1 Struct JuryGroupPending

addrJury	address	
tag	string	

```
3 struct JuryGroupPending {
4     address addrJury;
5     string tag;
6 }
```

4.1.3 Contract Definitions

- IBftgRoot

4.2 Module "IBftgRootStore.sol"

4.2.1 Pragmas

ton	-solidity >= 0.42.0	

4.2.2 Type Definitions

4.2.2.1 Enum ContractCode

JuryGroup	
Contest	

```
3  enum ContractCode {
4      JuryGroup,
5      Contest
6  }
```

4.2.2.2 Enum ContractAddr

empty	

```
8  enum ContractAddr {
9      empty
10 }
```

4.2.3 Contract Definitions

- IBftgRootStore
- IBftgRootStoreCallback

4.3 Module "IClient.sol"

4.3.1 Pragmas

ton	-solidity >= 0.36.0	

4.3.2 Imports

./IProposal.sol	
../Glossary.sol	

4.3.3 Contract Definitions

- IClient

4.4 Module "IGroup.sol"

4.4.1 Pragmas

ton	-solidity >= 0.36.0	

4.4.2 Contract Definitions

- IGroup
- IGroupCallback

4.5 Module "IJuryGroup.sol"

4.5.1 Pragmas

ton	-solidity >= 0.43.0	

4.5.2 Type Definitions

4.5.2.1 Struct Member

id	uint32	
balance	uint128	
addr	address	

```
3  struct Member {
4      uint32 id;
5      uint128 balance;
6      address addr;
7  }
```

4.5.3 Contract Definitions

- IJuryGroup
- IJuryGroupCallback

4.6 Module "IPadawan.sol"

4.6.1 Pragmas

ton	-solidity >= 0.36.0	

4.6.2 Imports

./IProposal.sol	

4.6.3 Type Definitions

4.6.3.1 Struct TipAccount

addr	address	
balance	uint128	

```
5  struct TipAccount {
6      address addr;
7      uint128 balance;
8  }
```

4.6.4 Contract Definitions

- IPadawan

4.7 Module "IProposal.sol"

4.7.1 Pragmas

ton	-solidity >= 0.36.0	

4.7.2 Imports

../Glossary.sol	

4.7.3 Type Definitions

4.7.3.1 Struct ProposalResults

id	uint32	
passed	bool	
votesFor	uint128	
votesAgainst	uint128	
totalVotes	uint256	
model	VoteCountModel	
ts	uint32	

```
5  struct ProposalResults {
6      uint32 id;
7      bool passed;
8      uint128 votesFor;
9      uint128 votesAgainst;
10     uint256 totalVotes;
11     VoteCountModel model;
12     uint32 ts;
13 }
```

4.7.3.2 Struct ProposalInfo

start	uint32	
end	uint32	
title	string	
proposalType	string	
specific	TvmCell	
state	ProposalState	
votesFor	uint128	
votesAgainst	uint128	
totalVotes	uint128	

```
15 struct ProposalInfo {
16     uint32 start;
17     uint32 end;
18     string title;
19     string proposalType;
```

```
20      TvmCell specific;
21      ProposalState state;
22      uint128 votesFor;
23      uint128 votesAgainst;
24      uint128 totalVotes;
25  }
```

4.7.4 Contract Definitions

- IProposal
- IEstimateVotesCallback

4.8 Module "ITokenRoot.sol"

4.8.1 Pragmas

ton	-solidity >= 0.42.0	

4.8.2 Contract Definitions

- ITokenRoot

4.9 Module "ITokenWallet.sol"

4.9.1 Pragmas

ton	-solidity >= 0.42.0	

4.9.2 Contract Definitions

- ITokenWallet

Chapter 5

Contract Modules

5.1 Module "Base.sol"

5.1.1 Pragmas

ton	-solidity >= 0.42.0	
msgValue	2e7	

5.1.2 Imports

./Errors.sol	

5.1.3 Contract Definitions

- Base

5.2 Module "BftgRoot.sol"

5.2.1 Pragmas

ton	-solidity >=0.36.0	
AbiHeader	expire	
AbiHeader	time	

5.2.2 Imports

./Base.sol	
./Checks.sol	
./Errors.sol	
./interfaces/IBftgRoot.sol	
./resolvers/ContestResolver.sol	
./resolvers/JuryGroupResolver.sol	

5.2.3 Contract Definitions

- BftgRoot

5.3 Module "Checks.sol"

5.3.1 Pragmas

ton	-solidity >= 0.42.0	

5.3.2 Contract Definitions

- Checks

5.4 Module "Contest.sol"

5.4.1 Pragmas

ton	-solidity >= 0.43.0	

5.4.2 Imports

./Checks.sol	
./interfaces/IContest.sol	
./interfaces/IBftgRoot.sol	
./interfaces/IBftgRootStore.sol	
./resolvers/JuryGroupResolver.sol	

5.4.3 Contract Definitions

- Contest

5.5 Module "ContestResolver.sol"

5.5.1 Pragmas

ton	-solidity >= 0.43.0	
AbiHeader	expire	
AbiHeader	time	

5.5.2 Imports

../Contest.sol	

5.5.3 Contract Definitions

- ContestResolver

5.6 Module "Group.sol"

5.6.1 Pragmas

ton	-solidity >= 0.36.0	
AbiHeader	expire	
AbiHeader	time	

5.6.2 Imports

./Base.sol	
./Errors.sol	
./interfaces/IGroup.sol	

5.6.3 Contract Definitions

- Group

5.7 Module "GroupResolver.sol"

5.7.1 Pragmas

ton	-solidity >= 0.36.0	
AbiHeader	expire	
AbiHeader	time	

5.7.2 Imports

../Group.sol	

5.7.3 Contract Definitions

- GroupResolver

5.8 Module "JuryGroup.sol"

5.8.1 Pragmas

ton	-solidity >= 0.36.0	

5.8.2 Imports

./interfaces/IJuryGroup.sol	

5.8.3 Contract Definitions

- JuryGroup

5.9 Module "JuryGroupResolver.sol"

5.9.1 Pragmas

ton	-solidity >= 0.42.0	

5.9.2 Imports

../JuryGroup.sol	

5.9.3 Contract Definitions

- JuryGroupResolver

5.10 Module "Padawan.sol"

5.10.1 Pragmas

ton	-solidity >= 0.36.0	
AbiHeader	expire	
AbiHeader	time	

5.10.2 Imports

./Base.sol	
./Errors.sol	
./interfaces/IProposal.sol	
./interfaces/IPadawan.sol	
./interfaces/ITokenRoot.sol	
./interfaces/ITokenWallet.sol	

5.10.3 Type Definitions

5.10.3.1 Struct PadawanData

ownerAddress	address	
addr	address	

```
12 struct PadawanData {
13     address ownerAddress;
14     address addr;
15 }
```

5.10.3.2 Struct Balance

total	uint128	
locked	uint128	

```
16 struct Balance {
17     uint128 total;
18     uint128 locked;
19 }
```

5.10.3.3 Struct ActiveProposal

voteProvider	address	
votePrice	uint128	
votes	uint128	

```
20 struct ActiveProposal {
21     address voteProvider;
22     uint128 votePrice;
```

```
23      uint128 votes;
24  }
```

5.10.3.4 Struct Reclaim

balanceProvider	address	
amount	uint128	
returnTo	address	

```
25  struct Reclaim {
26      address balanceProvider;
27      uint128 amount;
28      address returnTo;
29  }
```

5.10.4 Contract Definitions

- Padawan

5.11 Module "PadawanResolver.sol"

5.11.1 Pragmas

ton	-solidity >= 0.36.0	
AbiHeader	expire	
AbiHeader	time	

5.11.2 Imports

../Padawan.sol	

5.11.3 Contract Definitions

- PadawanResolver

5.12 Module "Proposal.sol"

5.12.1 Pragmas

ton	-solidity >= 0.36.0	
AbiHeader	expire	
AbiHeader	time	

5.12.2 Imports

./Base.sol	
./Errors.sol	
./resolvers/PadawanResolver.sol	
./resolvers/GroupResolver.sol	
./interfaces/IClient.sol	
./interfaces/IProposal.sol	
./interfaces/IPadawan.sol	
./interfaces/IGroup.sol	

5.12.3 Contract Definitions

- Proposal

Chapter 6

Contract Base

Contents

6.1 Overview

In file `Base.sol`

6.2 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;
```

6.3 Modifier Definitions

6.3.1 Modifier signed

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

6.3.2 Modifier accept

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

6.3.3 Modifier onlyContract

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

6.3.4 Modifier onlyMe

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

Chapter 7

Contract BftgRoot

Contents

7.1 Overview

In file `BftgRoot.sol`

7.2 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	

7.3 Constant Definitions

```
18     uint8 constant CHECK_CONTEST_CODE = 1;

19     uint8 constant CHECK_JURY_GROUP_CODE = 2;
```

7.4 Variable Definitions

```
34     address _addrBftgRootStore;

54     bool public _inited = false;

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

7.5 Modifier Definitions

7.5.1 Modifier onlyStore

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

7.6 Constructor Definitions

7.6.1 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      }
```

7.7 Public Method Definitions

7.7.1 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      }
```

7.7.2 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   }
```

7.7.3 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   }
```

7.7.4 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   }
```

7.7.5 Function getStored

- OK

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

7.7.6 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     }
```

7.7.7 Function updateAddr

- OK

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

7.7.8 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      }
```

7.8 Internal Method Definitions

7.8.1 Function _createChecks

- OK

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

7.8.2 Function _onInit

- OK

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

Chapter 8

Contract Checks

Contents

8.1 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.

8.2 Variable Definitions

```
4     uint8 _checkList;
```

8.3 Modifier Definitions

8.3.1 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      }
```

8.4 Internal Method Definitions

8.4.1 Function _isCheckListEmpty

- OK

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

8.4.2 Function _passCheck

- OK

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

Chapter 9

Contract Contest

Contents

9.1 Overview

In file `Contest.sol`

9.2 Contract Inheritance

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

JuryGroupResolver	
IJuryGroupCallback	
IBftgRootStoreCallback	
Checks	

9.3 Constant Definitions

- OK

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

9.4 Static Variable Definitions

- OK

```
28    address static _deployer;
```

9.5 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;
```

9.6 Constructor Definitions

9.6.1 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      }
```

9.7 Public Method Definitions

9.7.1 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      }
```

9.7.2 Function calcRewards

- **Critical issue: Not reentrant in `Contest.calcRewards`**
  This function is not reentrant, meaning that an attacker could call it several times to modify the rewards given to the participants. The function should:
  - Check that the voting stage is over, and that the reward stage is not yet started
  - Initialize the `_rewards` table to 0 before starting adding rewards to it (in case a computation was already done)

- **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     }
```

9.7.3 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     }
```

9.7.4 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     }
```

9.7.5 Function getHiddenVotesByAddress

- OK

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

9.7.6 Function getMembersCallback

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

```
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      }
```

9.7.7 Function hashVote

- TODO

```
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     }
```

9.7.8 Function reveal

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

```
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     }
```

9.7.9 Function stakePartisipantReward

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

```
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     }
```

9.7.10 Function submit

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

```
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     }
```

9.7.11 Function updateAddr

- TODO

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

9.7.12 Function updateCode

- TODO

```
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     }
```

9.7.13 Function vote

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

```
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    }
```

9.8 Internal Method Definitions

9.8.1 Function _calcPointValue

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

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

9.8.2 Function _changeStage

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

```
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     }
```

9.8.3 Function _createChecks

- TODO

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

9.8.4 Function _onInit

- TODO

```
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 10

Contract ContestResolver

Contents

10.1 Overview

In file `ContestResolver.sol`

10.2 Variable Definitions

TvmCell	_codeContest	
		assigned in @1.BftgRoot.updateCode
		used in @1.BftgRoot.updateCode
		used in @1.BftgRoot.getStored
		used in @8.ContestResolver._buildContestState

```
8     TvmCell _codeContest;
```

10.3 Public Method Definitions

10.3.1 Function resolveContest

- TODO

```
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      }
```

10.4 Internal Method Definitions

10.4.1 Function _buildContestState

- TODO

```
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 11

Contract Group

Contents

11.1 Overview

In file `Group.sol`

11.2 Contract Inheritance

Base	
IGroup	

11.3 Static Variable Definitions

string	_name	
		used in @21.Group.getMembers

```
11      string static _name;
```

11.4 Variable Definitions

address []	_members	
		assigned in @21.Group.removeMember
		used in @21.Group.removeMember
		used in @21.Group.removeMember
		used in @21.Group.removeMember
		used in @21.Group.removeMember
		used in @21.Group.getMembers
		used in @21.Group.addMember
		assigned in @21.Group.:constructor
		used in @21.Group.:constructor

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

11.5 Constructor Definitions

11.5.1 Constructor

- TODO

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

11.6 Public Method Definitions

11.6.1 Function addMember

- TODO

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

11.6.2 Function getMembers

- TODO

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

11.6.3 Function removeMember

- TODO

```
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 12

Contract GroupResolver

Contents

12.1 Overview

In file `GroupResolver.sol`

12.2 Variable Definitions

TvmCell	_codeGroup	
		used in @16.GroupResolver._buildGroupState

```
8     TvmCell _codeGroup;
```

12.3 Public Method Definitions

12.3.1 Function resolveGroup

- TODO

```
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      }
```

12.4 Internal Method Definitions

12.4.1 Function _buildGroupState

- TODO

```
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 13

Contract JuryGroup

Contents

13.1 Overview

In file `JuryGroup.sol`

13.2 Contract Inheritance

IJuryGroup	

13.3 Static Variable Definitions

string	_tag	
address	_deployer	
		used in @19.JuryGroup.:constructor

```
11     string static public _tag;
```

```
12     address static _deployer;
```

13.4 Variable Definitions

mapping (address => Member)	_members	
		assigned in @19.JuryGroup.withdraw
		used in @19.JuryGroup.withdraw
		used in @19.JuryGroup.withdraw
		used in @19.JuryGroup.withdraw
		assigned in @19.JuryGroup.registerMember
		used in @19.JuryGroup.registerMember
		used in @19.JuryGroup.registerMember
		used in @19.JuryGroup.getMembers
		assigned in @19.JuryGroup._addMember
		used in @19.JuryGroup._addMember
uint32	_membersCounter	
		assigned in @19.JuryGroup._addMember
		used in @19.JuryGroup._addMember
		used in @19.JuryGroup._addMember

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

```
15     uint32 _membersCounter;
```

13.5 Modifier Definitions

13.5.1 Modifier onlyDeployer

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

13.6 Constructor Definitions

13.6.1 Constructor

- TODO

```
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    }
```

13.7 Public Method Definitions

13.7.1 Function getMembers

- TODO

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

13.7.2 Function registerMember

- TODO

```
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    }
```

13.7.3 Function withdraw

- TODO

```
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      }
```

13.8 Internal Method Definitions

13.8.1 Function _addMember

- TODO

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

Chapter 14

Contract JuryGroupResolver

Contents

14.1 Overview

In file `JuryGroupResolver.sol`

14.2 Variable Definitions

TvmCell	_codeJuryGroup	
		assigned in @1.BftgRoot.updateCode
		used in @1.BftgRoot.updateCode
		used in @1.BftgRoot.getStored
		assigned in @18.Contest.updateCode
		used in @18.Contest.updateCode
		used in @9.JuryGroupResolver._buildJuryGroupState

```
6       TvmCell _codeJuryGroup;
```

14.3 Public Method Definitions

14.3.1 Function resolveJuryGroup

- TODO

```
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      }
```

14.4 Internal Method Definitions

14.4.1 Function _buildJuryGroupState

- TODO

```
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      }
```

Chapter 15

Contract Padawan

Contents

15.1 Overview

In file `Padawan.sol`

15.2 Contract Inheritance

Base	
IEstimateVotesCallback	

15.3 Static Variable Definitions

address	_deployer	
		used in @2.Padawan.:constructor
address	_owner	
		used in @2.Padawan.rejectVote
		used in @2.Padawan.onTokenWalletDeploy
		used in @2.Padawan.onEstimateVotes
		used in @2.Padawan.confirmVote
		used in @2.Padawan._doReclaim

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

15.4 Variable Definitions

mapping (address => Balance)	_balances	
		used in @2.Padawan.updateStatus
		assigned in @2.Padawan.updateStatus
		used in @2.Padawan.updateStatus
		used in @2.Padawan.updateStatus
		used in @2.Padawan.reclaimDeposit
		assigned in @2.Padawan.onTokenWalletGetBalance
		used in @2.Padawan.onTokenWalletGetBalance
		used in @2.Padawan.onTokenWalletGetBalance
		assigned in @2.Padawan.onTokenWalletDeploy
		used in @2.Padawan.onTokenWalletDeploy
		used in @2.Padawan.onEstimateVotes
		used in @2.Padawan.onEstimateVotes
		assigned in @2.Padawan.depositTons
		used in @2.Padawan.depositTons
		assigned in @2.Padawan.confirmVote
		used in @2.Padawan.confirmVote
		used in @2.Padawan.confirmVote
		assigned in @2.Padawan._doReclaim
		used in @2.Padawan._doReclaim
mapping (address => address)	_tokenAccounts	
		used in @2.Padawan.updateStatus
		used in @2.Padawan.reclaimDeposit
		assigned in @2.Padawan.onTokenWalletDeploy
		used in @2.Padawan.onTokenWalletDeploy
		used in @2.Padawan.onTokenWalletDeploy
		used in @2.Padawan.onEstimateVotes
		used in @2.Padawan.depositTokens

```
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;
```

15.5 Modifier Definitions

15.5.1 Modifier onlyOwner

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

15.6 Constructor Definitions

15.6.1 Constructor

- TODO

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

15.7 Public Method Definitions

15.7.1 Function confirmVote

- TODO

```
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     }
```

15.7.2 Function createTokenAccount

- TODO

```
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     }
```

15.7.3 Function depositTokens

- TODO

```
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     }
```

15.7.4 Function depositTons

- TODO

```
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     }
```

15.7.5 Function onEstimateVotes

- TODO

```
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      }
```

15.7.6 Function onTokenWalletDeploy

- TODO

```
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     }
```

15.7.7 Function onTokenWalletGetBalance

- TODO

```
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     }
```

15.7.8 Function reclaimDeposit

- TODO

```
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     }
```

15.7.9 Function rejectVote

- TODO

```
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     }
```

15.7.10 Function updateStatus

- TODO

```
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     }
```

15.7.11 Function vote

- TODO

```
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     }
```

15.8 Internal Method Definitions

15.8.1 Function _doReclaim

- TODO

```
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 16

Contract PadawanResolver

Contents

16.1 Overview

In file `PadawanResolver.sol`

16.2 Variable Definitions

TvmCell	_codePadawan	
		used in @3.Proposal._buildPadawanState
		assigned in @3.Proposal.:constructor
		used in @3.Proposal.:constructor
		used in @17.PadawanResolver._buildPadawanState

```
8     TvmCell _codePadawan;
```

16.3 Public Method Definitions

16.3.1 Function resolvePadawan

- TODO

```
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      }
```

16.4 Internal Method Definitions

16.4.1 Function _buildPadawanState

- TODO

```
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 17

Contract Proposal

Contents

17.1 Overview

In file `Proposal.sol`

17.2 Contract Inheritance

Base	
PadawanResolver	
GroupResolver	
IProposal	
IGroupCallback	

17.3 Static Variable Definitions

address	_deployer	
		used in @3.Proposal._buildPadawanState
		used in @3.Proposal.:constructor
uint32	_id	

```
15     address static _deployer;

16     uint32 static _id;
```

17.4 Variable Definitions

address	_client	
		used in @3.Proposal._finalize
		assigned in @3.Proposal.:constructor
		used in @3.Proposal.:constructor
uint128	_votePrice	
		used in @3.Proposal.vote
		used in @3.Proposal.estimateVotes
		used in @3.Proposal.estimateVotes
		assigned in @3.Proposal.:constructor
		used in @3.Proposal.:constructor
uint128	_voteTotal	
		used in @3.Proposal._tryEarlyComplete
		used in @3.Proposal._tryEarlyComplete
		used in @3.Proposal._softMajority
		used in @3.Proposal._softMajority
		used in @3.Proposal._softMajority
		used in @3.Proposal._softMajority
		used in @3.Proposal._finalize
		assigned in @3.Proposal.:constructor
		used in @3.Proposal.:constructor
address	_voteProvider	
		used in @3.Proposal.vote
		used in @3.Proposal.estimateVotes
		assigned in @3.Proposal.:constructor
		used in @3.Proposal.:constructor
address []	_whiteList	
		assigned in @3.Proposal.onGetMembers
		used in @3.Proposal.onGetMembers
		used in @3.Proposal._findInWhiteList
		used in @3.Proposal._findInWhiteList
		assigned in @3.Proposal.:constructor
		used in @3.Proposal.:constructor
bool	_openProposal	Initialized to false
		used in @3.Proposal.vote

```
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;
```

17.5 Constructor Definitions

17.5.1 Constructor

- TODO

```
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      }
```

17.6 Public Method Definitions

17.6.1 Function estimateVotes

- TODO

```
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      }
```

17.6.2 Function getAll

- TODO

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

17.6.3 Function getCurrentVotes

- TODO

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

17.6.4 Function getInfo

- TODO

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

17.6.5 Function getVotingResults

- TODO

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

17.6.6 Function onGetMembers

- TODO

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

17.6.7 Function queryStatus

- TODO

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

17.6.8 Function vote

- TODO

```
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     }
```

17.6.9 Function wrapUp

- TODO

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

17.7 Internal Method Definitions

17.7.1 Function _buildPadawanState

- TODO

```
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     }
```

17.7.2 Function _calculateVotes

- TODO

```
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     }
```

17.7.3 Function _changeState

- TODO

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

17.7.4 Function _finalize

- TODO

```
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     }
```

17.7.5 Function _findInWhiteList

- TODO

```
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     }
```

17.7.6 Function _getGroupMembers

- TODO

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

17.7.7 Function _softMajority

- TODO

```
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     }
```

17.7.8 Function _tryEarlyComplete

- TODO

```
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     }
```

17.7.9 Function _wrapUp

- TODO

```
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     }
```