Audit of the BFTG project

By OCamlPro

August 20, 2021

Table of Major and Critical Issues

Contents

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	

```
enum VoteCountModel {
    Undefined,
    Majority,
    SoftMajority,
    SuperMajority,
    Other,
    Reserved,
    Last
}
```

3.3.2.2 Enum ProposalType

Undefined	
SetCode	
Reserve	
SetOwner	
SetRootOwner	

```
enum ProposalType {
    Undefined,
    SetCode,
    Reserve,
    SetOwner,
    SetRootOwner
}
```

3.3.2.3 Enum ProposalState

Undefined	
New	
OnVoting	
Ended	
Passed	
NotPassed	
Finalized	
Distributed	
Reserved	
Last	

```
enum ProposalState {
    Undefined,
    New,
    OnVoting,
    Ended,
    Passed,
    NotPassed,
    Finalized,
    Distributed,
    Reserved,
    Last
}
```

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	

```
enum ContestStage {
    Undefined,
    New,
    Underway,
    Voting,
    Reveal,
    Rank,
    Reward,
    Finish,
    Last
}
```

3.4.2.2 Struct Submission

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

```
struct Submission {
  uint32 id;
  address addrPartisipant;
  string forumLink;
  string fileLink;
  uint hash;
  uint32 createdAt;
}
```

3.4.2.3 Struct HiddenVote

submissionId	uint32	
hash	uint256	
hiddenComment	bytes	
hiddenScore	bytes	

```
struct HiddenVote {
    uint32 submissionId;
    uint hash;
    bytes hiddenComment;
    bytes hiddenScore;
}
```

3.4.2.4 Struct RevealVote

submissionId	uint32	
score	uint8	
comment	bytes	

```
struct RevealVote {
    uint32 submissionId;
    uint8 score;
    bytes comment;
}
```

3.4.2.5 Struct Vote

addrJury	address	
score	uint8	
comment	bytes	

```
struct Vote {
    address addrJury;
    uint8 score;
    bytes comment;
}
```

3.4.2.6 Struct Reward

total	uint128	
paid	uint128	

```
struct Reward {
    uint128 total;
    uint128 paid;
}
```

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	

```
struct JuryGroupPending {
    address addrJury;
    string tag;
}
```

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	

```
enum ContractCode {
    JuryGroup,
    Contest
}
```

4.2.2.2 Enum ContractAddr

empty	

```
enum ContractAddr {
    empty
}
```

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	

```
struct Member {
    uint32 id;
    uint128 balance;
    address addr;
}
```

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	

```
struct TipAccount {
    address addr;
    uint128 balance;
}
```

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	

```
struct ProposalResults {
    uint32 id;
    bool passed;
    uint128 votesFor;
    uint128 votesAgainst;
    uint256 totalVotes;
    VoteCountModel model;
    uint32 ts;
}
```

4.7.3.2 Struct ProposalInfo

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

```
struct ProposalInfo {
    uint32 start;
    uint32 end;
    string title;
    string proposalType;
    TvmCell specific;
    ProposalState state;
    uint128 votesFor;
    uint128 votesAgainst;
    uint128 totalVotes;
}
```

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	

```
struct PadawanData {
    address ownerAddress;
    address addr;
}
```

5.10.3.2 Struct Balance

total	uint128	
locked	uint128	

```
struct Balance {
    uint128 total;
    uint128 locked;
}
```

5.10.3.3 Struct ActiveProposal

voteProvider	address	
votePrice	uint128	
votes	uint128	

```
struct ActiveProposal {
    address voteProvider;
    uint128 votePrice;
    uint128 votes;
}
```

5.10.3.4 Struct Reclaim

balanceProvider	address	
amount	uint128	
returnTo	address	

```
struct Reclaim {
    address balanceProvider;
    uint128 amount;
    address returnTo;
}
```

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

```
    uint64 constant DEFAULT_FEE          = 1 ton;

    uint16 constant ERROR_DIFFERENT_CALLER =  211;

    uint64 constant START_BALANCE        = 3 ton;
    uint64 constant DEPLOYER_FEE         = 0.1 ton;
    uint64 constant PROCESS_FEE          = 0.3 ton;
    uint64 constant VOTE_FEE             = 1 ton;
    uint64 constant DEPLOY_FEE           = START_BALANCE +
        DEPLOYER_FEE;
    uint64 constant DEPLOY_PAY          = DEPLOY_FEE + PROCESS_FEE;
    uint64 constant DEPLOY_PROPOSAL_FEE = 3 ton;
    uint64 constant DEPLOY_PROPOSAL_PAY = DEPLOY_PROPOSAL_FEE +
        PROCESS_FEE;
    uint64 constant DEPOSIT_TONS_FEE    = 1 ton;
    uint64 constant DEPOSIT_TONS_PAY    = DEPOSIT_TONS_FEE +
        PROCESS_FEE;
    uint64 constant DEPOSIT_TOKENS_FEE  = 0.5 ton +
        DEPOSIT_TONS_FEE;
    uint64 constant DEPOSIT_TOKENS_PAY  = DEPOSIT_TOKENS_FEE +
        PROCESS_FEE;
    uint64 constant TOKEN_ACCOUNT_FEE   = 2 ton;
    uint64 constant TOKEN_ACCOUNT_PAY   = TOKEN_ACCOUNT_FEE +
        PROCESS_FEE;
    uint64 constant QUERY_STATUS_FEE    = 0.2 ton;
    uint64 constant QUERY_STATUS_PAY    = QUERY_STATUS_FEE +
        DEF_RESPONSE_VALUE;
    uint64 constant DEF_RESPONSE_VALUE = 0.03 ton;
    uint64 constant DEF_COMPUTE_VALUE = 0.2 ton;
```

6.3 Modifier Definitions

6.3.1 Modifier signed

```
    modifier signed {
        require(msg.pubkey() == tvm.pubkey(), 100);
        tvm.accept();
        _;
    }
```

6.3.2 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.

```
    modifier accept {
        tvm.accept();
        _;
    }
```

6.3.3 Modifier onlyContract

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

6.3.4 Modifier onlyMe

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

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

```
    uint8 constant CHECK_CONTEST_CODE = 1;
```

```
    uint8 constant CHECK_JURY_GROUP_CODE = 2;
```

7.4 Variable Definitions

```
    address _addrBftgRootStore;
```

```
    bool public _inited = false;
```

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

7.5 Modifier Definitions

7.5.1 Modifier onlyStore

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

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.

```
    constructor(address addrBftgRootStore) public {
        if (msg.sender == address(0)) {
            require(msg.pubkey() == tvm.pubkey(), Errors.
                ONLY_SIGNED);
        }
        require(addrBftgRootStore != address(0), Errors.
            STORE_UNDEFINED);
        tvm.accept();

        _addrBftgRootStore = addrBftgRootStore;
        IBftgRootStore(addrBftgRootStore).queryCode
            {value: 0.2 ton, bounce: true}
            (ContractCode.Contest);
        IBftgRootStore(addrBftgRootStore).queryCode
            {value: 0.2 ton, bounce: true}
            (ContractCode.JuryGroup);

        _createChecks();
    }
```

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.

```
    onBounce(TvmSlice) external {
        if(_juryGroupPendings.exists(msg.sender)) {
            address[] _;
            deployJuryGroup(_juryGroupPendings[msg.sender].tag, _);
            this.registerMemberJuryGroup
                {value: 0, bounce: false, flag: 64}
                (_juryGroupPendings[msg.sender].tag,
                    _juryGroupPendings[msg.sender].addrJury);
            delete _juryGroupPendings[msg.sender];
        }
    }
```

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.

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

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.

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

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.

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

7.7.5 Function getStored

- OK

```
    function getStored() public view returns (
        TvmCell codeContest,
        TvmCell codeJuryGroup
    ) {
        codeContest = _codeContest;
        codeJuryGroup = _codeJuryGroup;
    }
```

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.

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

7.7.7 Function updateAddr

- OK

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

7.7.8 Function updateCode

- OK

```
    function updateCode(
        ContractCode kind,
        TvmCell code
    ) external override onlyStore {
        if (kind == ContractCode.Contest) {
            _codeContest = code;
            _passCheck(CHECK_CONTEST_CODE);
        }
        if (kind == ContractCode.JuryGroup) {
            _codeJuryGroup = code;
            _passCheck(CHECK_JURY_GROUP_CODE);
        }
        _onInit();
    }
```

7.8 Internal Method Definitions

7.8.1 Function _createChecks

- OK

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

7.8.2 Function _onInit

- OK

```
    function _onInit() private {
        if(_isCheckListEmpty() && !_inited) {
            _inited = true;
        }
    }
```

Chapter 8

Contract BftgRootStore

Contents

8.1 Overview

In file `BftgRootStore.sol`

8.2 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).

8.3 Contract Inheritance

Base	
IBftgRootStore	

8.4 Variable Definitions

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

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

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

8.5 Public Method Definitions

8.5.1 Function queryAddr

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

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

8.5.2 Function queryCode

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

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

8.5.3 Function setContestCode

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

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

8.5.4 Function setJuryGroupCode

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

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

Chapter 9

Contract Checks

Contents

9.1 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.

9.2 Variable Definitions

```
    uint8 _checkList;
```

9.3 Modifier Definitions

9.3.1 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`.

```
    modifier checksEmpty() {
        require(_isCheckListEmpty(), 100); //Errors.NOT_ALL_CHECKS_PASSED);
        tvm.accept();
        _;
    }
```

9.4 Internal Method Definitions

9.4.1 Function _isCheckListEmpty

- OK

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

9.4.2 Function _passCheck

- OK

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

Chapter 10

Contract Contest

Contents

10.1 Overview

In file Contest.sol

10.2 Contract Inheritance

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

JuryGroupResolver	
IJuryGroupCallback	
IBftgRootStoreCallback	
Checks	

10.3 Constant Definitions

- OK

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

10.4 Static Variable Definitions

- OK

```
  address static _deployer;
```

10.5 Variable Definitions

- OK

```
  string[] public _tags;
```

```
  mapping(address => bool) _tagsPendings;
```

```
  mapping(address => Member) public _jury;
```

```
  uint128 public _prizePool;
```

```
  uint32 public _underwayDuration;
```

```
  uint32 public _underwayEnds;
```

```
  bool public _inited = false;
```

```
    ContestStage public _stage;
```

```
    mapping(uint32 => Submission) public _submissions;
```

```
    uint32 _submissionsCounter;
```

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

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

```
    uint128 _pointValue;
```

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

10.6 Constructor Definitions

10.6.1 Constructor

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

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

10.7 Public Method Definitions

10.7.1 OnBounce function

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

```
    onBounce(TvmSlice) external {
        if(_tagsPendings.exists(msg.sender)) {
            delete _tagsPendings[msg.sender];
            if(_tagsPendings.empty()) {
                _changeStage(ContestStage.Underway);
            }
        }
    }
```

10.7.2 Function calcRewards

- **Critical issue: No stage check in `Contest.calcRewards`**

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

- **Major issue: Wrong computation in `Contest.calcRewards`**

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

```
    function calcRewards() public {
        _calcPointValue();
        optional(uint32, Vote[]) optSubmissionVotes =
            _submissionVotes.min();
        while (optSubmissionVotes.hasValue()) {
            (uint32 id, Vote[] submissionVotes) =
                optSubmissionVotes.get();
            for(uint8 i = 0; i < submissionVotes.length; i++) {
                _rewards[_submissions[id].addrPartisipant].total +=
                     submissionVotes[i].score * _pointValue;
            }
            optSubmissionVotes = _submissionVotes.next(id);
        }
        _changeStage(ContestStage.Reward);
    }
```

10.7.3 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`.

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

10.7.4 Function claimPartisipantReward

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

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

10.7.5 Function getHiddenVotesByAddress

- OK

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

10.7.6 Function getMembersCallback

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

- Minor issue: the test `member.balance >= 0` is useless as the field is an unsigned integer `uint128`.

```
    function getMembersCallback(mapping(address => Member) members) external override {
        require(_tagsPendings.exists(msg.sender), 102);
        delete _tagsPendings[msg.sender];
        for((, Member member): members) {
            if(member.balance >= 0) {
                _jury[member.addr] = member;
            }
        }
        if(_tagsPendings.empty()) {
            _changeStage(ContestStage.Underway);
        }
    }
```

10.7.7 Function hashVote

- OK

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

10.7.8 Function reveal

- **Critical issue: Multiple revelations in `Contest.reveal`**
  A jury can reveal his votes several times, adding them several times in the `_submissionVotes` table. Fix: remove submission from `_juryHiddenVotes` everytime they are revealed.

- Minor issue (gas cost): instead of failing if `oldHash` and `newHash` differ, the function should probably returns the list of failed couples, and keep working for correct couples.

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

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

10.7.9 Function stakePartisipantReward

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

```
    function stakePartisipantReward(uint128 amount, string tag,
        address addrJury) public {
        require(_rewards.exists(msg.sender), 107);
        require(_rewards[msg.sender].total - _rewards[msg.sender].
            paid >= amount, 108);
        bool isTagExists = false;
        for(uint8 i = 0; i < _tags.length; i++) {
            if(_tags[i] == tag) isTagExists = true;
        }
        require(isTagExists, 108);
        _rewards[msg.sender].paid += amount;
        IBftgRoot(_deployer).registerMemberJuryGroup
            {value: amount, bounce: true, flag: 2}
            (tag, addrJury == address(0) ? msg.sender : addrJury);
        msg.sender.transfer(0, true, 64);
    }
```

10.7.10 Function submit

- **Major issue: Unbounded storage in `Contest.submit`**

  Anybody can call this function. An attacker could use it to increase dramatically the cost of calling the contract by storing a very big submission into the contest storage.

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

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

10.7.11 Function updateAddr

- OK

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

10.7.12 Function updateCode

- **Critical issue: No permission check in `Contest.updateCode`**

  No check is performed on the sender of this message, allowing an attacker to provide his own malicious implementation of `JuryGroup` to the contract. Fix: check the sender, or check the code hash of the code.

- **Major issue: No gas check in `Contest.updateCode`**

  Given that this function is responsible for sending `getMembers` messages to all jury groups, it should check by `require` that the message contains enough gas to perform these sends. Otherwise, it could happen that the action phase could succeed, the contract would remember that it was initialized, yet the transaction would be aborted in the sending phase and no message would actually be sent by lack of gas.

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

- Minor issue: if `kind` is not `ContractCode.JuryGroup`, this function will silently return without error, whereas the user might interpret it as successful and initialization done. Fix: replace the `if` by a `require`.

```
    function updateCode(ContractCode kind, TvmCell code) external
        override {
        if (kind == ContractCode.JuryGroup) {
            _codeJuryGroup = code;
            _passCheck(CHECK_JURY_GROUP_CODE);
        }
        _onInit();
    }
```

10.7.13 Function vote

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

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

10.8 Internal Method Definitions

10.8.1 Function _calcPointValue

- OK

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

10.8.2 Function _changeStage

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

```
    function _changeStage(ContestStage stage) private inline
        returns (ContestStage) {
        require(_stage < stage, 103);
        if (stage == ContestStage.Underway) {
            _underwayEnds = uint32(now) + _underwayDuration;
        }
        _stage = stage;
    }
```

10.8.3 Function _createChecks

- OK

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

10.8.4 Function _onInit

- TODO

```
    function _onInit() private {
        if(_isCheckListEmpty() && !_inited) {
            _inited = true;
            for(uint8 i = 0; i < _tags.length; i++) {
                TvmCell state = _buildJuryGroupState(_tags[i],
                    _deployer);
                uint256 hashState = tvm.hash(state);
                address addrJuryGroup = address.makeAddrStd(0,
                    hashState);
                _tagsPendings[addrJuryGroup] = true;
                IJuryGroup(addrJuryGroup).getMembers{
                    value: 0.2 ton,
                    flag: 1,
                    bounce: true
                }();
            }
        }
    }
```

Chapter 11

Contract ContestResolver

Contents

11.1 Overview

In file `ContestResolver.sol`

11.2 Variable Definitions

```
    TvmCell _codeContest;
```

11.3 Public Method Definitions

11.3.1 Function resolveContest

- OK

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

11.4 Internal Method Definitions

11.4.1 Function _buildContestState

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

```
    function _buildContestState(address deployer) internal virtual view returns (TvmCell) {
        return tvm.buildStateInit({
            contr: Contest,
            varInit: {_deployer: deployer},
            code: _codeContest
        });
    }
```

Chapter 12

Contract Group

Contents

12.1 Overview

In file `Group.sol`

12.2 Contract Inheritance

Base	
IGroup	

12.3 Static Variable Definitions

```
    string static _name;
```

12.4 Variable Definitions

```
    address[] _members;
```

12.5 Constructor Definitions

12.5.1 Constructor

- **Critical issue: No permission check in `Group.constructor`**

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

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

12.6 Public Method Definitions

12.6.1 Function addMember

- **Critical issue: No permission check in `Group.addMember`**

  An attacker could add any member to the group because no permission check is performed in this function

- Minor issue: a member can be added several times in the group. Fix: use a mapping and only add non-existing members.

- Minor issue: the argument `idProposal` is not used.

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

12.6.2 Function getMembers

- OK

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

12.6.3 Function removeMember

- **Critical issue: No permission check on `removeMember`**

  An attacker could remove any member of the group, as no permission check is performed.

- Minor issue: the argument `idProposal` is not used.

```
    function removeMember(uint128 idProposal, address member)
        public onlyContract {
        idProposal;
        address[] members;
        for(uint32 index = 0; index < _members.length; index++) {
            if(_members[index] != member) {
                members.push(_members[index]);
            }
        }
        _members = members;
    }
```

Chapter 13

Contract GroupResolver

Contents

13.1 Overview

In file `GroupResolver.sol`

13.2 Variable Definitions

TvmCell	_codeGroup	
		used in @16.GroupResolver._buildGroupState

```
    TvmCell _codeGroup;
```

13.3 Public Method Definitions

13.3.1 Function resolveGroup

- OK

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

13.4 Internal Method Definitions

13.4.1 Function _buildGroupState

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

```
    function _buildGroupState(string name) internal virtual view returns (TvmCell) {
        return tvm.buildStateInit({
            contr: Group,
            varInit: {_name: name},
            code: _codeGroup
        });
    }
```

Chapter 14

Contract JuryGroup

Contents

14.1 Overview

In file `JuryGroup.sol`

14.2 Contract Inheritance

IJuryGroup	

14.3 Static Variable Definitions

```
    string static public _tag;
```

```
    address static _deployer;
```

14.4 Variable Definitions

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

```
    uint32 _membersCounter;
```

14.5 Modifier Definitions

14.5.1 Modifier onlyDeployer

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

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

14.6 Constructor Definitions

14.6.1 Constructor

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

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

14.7 Public Method Definitions

14.7.1 Function getMembers

- OK

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

14.7.2 Function registerMember

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

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

14.7.3 Function withdraw

- **Major issue: Wrong comparison in `JuryGroup.withdraw`**

  The check `_members[msg.sender].balance < amount` will fail, or if it does not fail, the operation `_members[msg.sender].balance -= amount` will fail. Either way, the function will always fail.

- Minor issue: the check `_members[msg.sender].balance >= 0 ton` is always true, because `balance` is an `uint128`.

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

14.8 Internal Method Definitions

14.8.1 Function _addMember

- OK

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

Chapter 15

Contract JuryGroupResolver

Contents

15.1 Overview

In file `JuryGroupResolver.sol`

15.2 Variable Definitions

```
    TvmCell _codeJuryGroup;
```

15.3 Public Method Definitions

15.3.1 Function resolveJuryGroup

- OK

```
    function resolveJuryGroup(string tag, address deployer) public
        view returns (address addrJuryGroup) {
        TvmCell state = _buildJuryGroupState(tag, deployer);
        uint256 hashState = tvm.hash(state);
        addrJuryGroup = address.makeAddrStd(0, hashState);
    }
```

15.4 Internal Method Definitions

15.4.1 Function _buildJuryGroupState

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

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

Chapter 16

Contract Padawan

Contents

16.1 Overview

In file `Padawan.sol`

16.2 Contract Inheritance

Base	
IEstimateVotesCallback	

16.3 Static Variable Definitions

- OK

```
    address static _deployer;
```

```
    address static _owner;
```

16.4 Variable Definitions

- OK

```
    mapping(address => Balance) public _balances;
```

```
    mapping(address => address) public _tokenAccounts;
```

```
    mapping(address => ActiveProposal) public _activeProposals;
```

```
    uint32 _activeProposalsLength;
```

```
    Reclaim public _reclaim;
```

16.5 Modifier Definitions

16.5.1 Modifier onlyOwner

- OK

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

16.6 Constructor Definitions

16.6.1 Constructor

- OK

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

16.7 Public Method Definitions

16.7.1 Function confirmVote

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

```
    function confirmVote(
        uint128 votes,
        uint128 votePrice,
        address voteProvider)
    external onlyContract { votes;
        optional(ActiveProposal) optActiveProposal =
            _activeProposals.fetch(msg.sender);
        require(optActiveProposal.hasValue(), 111);
        uint128 activeProposalVotes = optActiveProposal.get().votes
            ;

        address balanceProvider = voteProvider == address(0) ?
            voteProvider : _tokenAccounts[voteProvider];

        if(_balances[balanceProvider].locked < (activeProposalVotes
            ) * votePrice) {
            _balances[balanceProvider].locked = (
                activeProposalVotes) * votePrice;
        }
        _owner.transfer(0, false, 64);
    }
```

16.7.2 Function createTokenAccount

- OK

```
    function createTokenAccount(address tokenRoot) external
        onlyOwner {
        require(msg.value >= DEFAULT_FEE, Errors.MSG_VALUE_TOO_LOW)
            ;
        require(!_tokenAccounts.exists(tokenRoot));

        ITokenRoot(tokenRoot).deployEmptyWallet
```

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

16.7.3 Function depositTokens

- OK

```
    function depositTokens(address tokenRoot) external onlyOwner {
        require(msg.value >= DEFAULT_FEE, Errors.MSG_VALUE_TOO_LOW);
        optional(address) optTokenAccount = _tokenAccounts.fetch(tokenRoot);
        require(optTokenAccount.hasValue(), Errors.ACCOUNT_DOES_NOT_EXIST);

        address tokenAccount = optTokenAccount.get();

        ITokenWallet(tokenAccount).getBalance_InternalOwner
            {value: 0, flag: 64, bounce: true}
            (tvm.functionId(onTokenWalletGetBalance));
    }
```

16.7.4 Function depositTons

- OK

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

16.7.5 Function onEstimateVotes

- **Major issue: Incorrect computation in `Padawan.onEstimateVotes`**

  The value of `_activeProposalsLength` is wrong if the user sends his votes in multiple batches. Indeed, if this variable measures the size of the mapping `_activeProposals`, it should only be increased in the case `!optActiveProposal.hasValue()`. Otherwise, the value is increased for every batch of votes, and only decreased when all votes have been confirmed/rejected, leading to a over-estimation of the number of entries in the mapping.

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

```
    function onEstimateVotes(
        uint128 cost,
        uint128 votePrice,
        address voteProvider,
        uint128 votes,
        bool choice)
    external override onlyContract {
        optional(ActiveProposal) optActiveProposal =
            _activeProposals.fetch(msg.sender);
        ActiveProposal activeProposal = optActiveProposal.hasValue
            () ? optActiveProposal.get() : ActiveProposal(
            voteProvider, votePrice, 0);
        if(!optActiveProposal.hasValue()) {
            _activeProposals[msg.sender] = activeProposal;
        }
        optional(Balance) optBalance;
        if(voteProvider == address(0)) {
            optBalance = _balances.fetch(voteProvider);
        } else {
            optional(address) optAccount = _tokenAccounts.fetch(
                voteProvider);
            require(optAccount.hasValue(), 115);
            optBalance = _balances.fetch(optAccount.get());
        }
        require(optBalance.hasValue(), 113);
        require(optBalance.get().total >= (activeProposal.votes *
            votePrice) + cost, 114);
        _activeProposals[msg.sender].votes += votes;
        _activeProposalsLength += 1;
        IProposal(msg.sender).vote
            {value: 0, flag: 64, bounce: true}
            (_owner, choice, votes);
    }
```

16.7.6 Function onTokenWalletDeploy

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

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

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

16.7.7 Function onTokenWalletGetBalance

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

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

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

16.7.8 Function reclaimDeposit

- **Critical issue: Race condition in `Padawan.reclaimDeposit`**

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

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

```
function reclaimDeposit(address voteProvider, uint128 amount,
    address returnTo) external onlyOwner {
    require(_reclaim.amount == 0, 130);
    require(msg.value >= QUERY_STATUS_FEE *
        _activeProposalsLength + 1 ton, Errors.
        MSG_VALUE_TOO_LOW);
    address balanceProvider = address(0);
    if(voteProvider != address(0)) {
        optional(address) optAccount = _tokenAccounts.fetch(
            voteProvider);
        require(optAccount.hasValue(), 117);
        balanceProvider = optAccount.get();
    }
    optional(Balance) optBalance = _balances.fetch(
        balanceProvider);
    require(optBalance.hasValue(), 131);
    Balance balance = optBalance.get();
    require(amount <= balance.total, Errors.NOT_ENOUGH_VOTES);
    require(returnTo != address(0), 132);

    _reclaim = Reclaim(balanceProvider, amount, returnTo);

```

```
        if (amount <= balance.total - balance.locked) {
            _doReclaim();
        }

        optional(address, ActiveProposal) optActiveProposal =
            _activeProposals.min();
        while (optActiveProposal.hasValue()) {
            (address addrActiveProposal,) = optActiveProposal.get()
                ;
            IProposal(addrActiveProposal).queryStatus
                {value: QUERY_STATUS_FEE, bounce: true, flag: 1}
                ();
            optActiveProposal = _activeProposals.next(
                addrActiveProposal);
        }
    }
```

16.7.9 Function rejectVote

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

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

16.7.10 Function updateStatus

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

```
    function updateStatus(ProposalState state) external onlyContract {
        optional(ActiveProposal) optActiveProposal = _activeProposals.fetch(msg.sender);
        require(optActiveProposal.hasValue());
        ActiveProposal activeProposal = optActiveProposal.get();

        if (state >= ProposalState.Ended) {
            address balanceProvider = address(0);
            if(activeProposal.voteProvider != address(0)) {
                optional(address) optAccount = _tokenAccounts.fetch(activeProposal.voteProvider);
                require(optAccount.hasValue(), 117);
                balanceProvider = optAccount.get();
            }
            Balance balance = _balances[balanceProvider];
            if(balance.locked <= activeProposal.votes * activeProposal.votePrice) {
                delete _activeProposals[msg.sender];
                uint128 max;
                optional(address, ActiveProposal) optActiveProposal2 = _activeProposals.min();
                while (optActiveProposal2.hasValue()) {
                    (address addrActiveProposal, ActiveProposal activeProposal2) = optActiveProposal2.get();
                    if(activeProposal2.votes * activeProposal2.votePrice > max && activeProposal2.voteProvider == activeProposal.voteProvider) {
                        max = activeProposal2.votes * activeProposal2.votePrice;
                    }
                    optActiveProposal2 = _activeProposals.next(addrActiveProposal);
                }
                _balances[balanceProvider].locked = max;
            } else {
                delete _activeProposals[msg.sender];
            }
            _activeProposalsLength -= 1;
            if(_reclaim.amount != 0) {
                balance = _balances[_reclaim.balanceProvider];
                if (_reclaim.amount <= balance.total - balance.locked) {
                    _doReclaim();
                }
            }
        }
    }
```

16.7.11 Function vote

- OK

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

16.8 Internal Method Definitions

16.8.1 Function _doReclaim

- OK

```
    function _doReclaim() private inline {
        if(_reclaim.balanceProvider == address(0)) {
            _reclaim.returnTo.transfer(_reclaim.amount, true, 1);
        } else {
            ITokenWallet(_reclaim.balanceProvider).transfer
                {value: 0.2 ton} // refactor
                (_reclaim.returnTo, _reclaim.amount, 0.1 ton);
        }
        _balances[_reclaim.balanceProvider].total -= _reclaim.
            amount;
        delete _reclaim;
        _owner.transfer(0, false, 64);
    }
```

Chapter 17

Contract PadawanResolver

Contents

17.1 Overview

In file `PadawanResolver.sol`

17.2 Variable Definitions

```
    TvmCell _codePadawan;
```

17.3 Public Method Definitions

17.3.1 Function resolvePadawan

- OK

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

17.4 Internal Method Definitions

17.4.1 Function _buildPadawanState

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

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

Chapter 18

Contract Proposal

Contents

18.1 Overview

In file `Proposal.sol`

18.2 Contract Inheritance

Base	
PadawanResolver	
GroupResolver	
IProposal	
IGroupCallback	

18.3 Static Variable Definitions

- OK

```
    address static _deployer;

    uint32 static _id;
```

18.4 Variable Definitions

- OK

```
    address _client;

    uint128 _votePrice;

    uint128 _voteTotal;

    address _voteProvider;

    address[] _whiteList;

    bool _openProposal = false;

    ProposalInfo _proposalInfo;

    ProposalResults _results;

    VoteCountModel _voteCountModel;
```

18.5 Constructor Definitions

18.5.1 Constructor

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

```
    constructor(
        address client,
        string title,
        uint128 votePrice,
        uint128 voteTotal,
        address voteProvider,
        address group,
        address[] whiteList,
        string proposalType,
        TvmCell specific,
        TvmCell codePadawan
    ) public {
        require(_deployer == msg.sender);

        _client = client;

        _votePrice = votePrice;
        _voteTotal = voteTotal;
        _voteProvider = voteProvider;

        _proposalInfo.title = title;
        _proposalInfo.start = uint32(now);
        _proposalInfo.end = uint32(now + 60 * 60 * 24 * 7);
        _proposalInfo.proposalType = proposalType;
        _proposalInfo.specific = specific;
        _proposalInfo.state = ProposalState.New;
        _proposalInfo.totalVotes = voteTotal;

        _codePadawan = codePadawan;

        if(group != address(0)) {
            _getGroupMembers(group);
        } else if (!whiteList.empty()) {
            _whiteList = whiteList;
        } else  {
            _openProposal = true;
        }

        _voteCountModel = VoteCountModel.SoftMajority;
    }
```

18.6 Public Method Definitions

18.6.1 Function estimateVotes

- OK

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

18.6.2 Function getAll

- OK

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

18.6.3 Function getCurrentVotes

- OK

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

18.6.4 Function getInfo

- OK

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

18.6.5 Function getVotingResults

- OK

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

18.6.6 Function onGetMembers

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

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

18.6.7 Function queryStatus

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

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

18.6.8 Function vote

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

```
function vote(address padawanOwner, bool choice, uint128 votes)
     external override {
    address addrPadawan = resolvePadawan(padawanOwner);
    uint16 errorCode = 0;

    require(_openProposal || _findInWhiteList(padawanOwner),
        Errors.INVALID_CALLER);

    if (addrPadawan != msg.sender) {
        errorCode = Errors.NOT_AUTHORIZED_CONTRACT;
    } else if (now < _proposalInfo.start) {
        errorCode = Errors.VOTING_NOT_STARTED;
    } else if (now > _proposalInfo.end) {
        errorCode = Errors.VOTING_HAS_ENDED;
    }

    if (errorCode > 0) {
        IPadawan(msg.sender).rejectVote{value: 0, flag: 64,
            bounce: true}(votes, errorCode);
    } else {
        IPadawan(msg.sender).confirmVote{value: 0, flag: 64,
            bounce: true}(votes, _votePrice, _voteProvider);
        if (choice) {
            _proposalInfo.votesFor += votes;
        } else {
            _proposalInfo.votesAgainst += votes;
        }
    }

    _wrapUp();
}
```

18.6.9 Function wrapUp

- OK

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

18.7 Internal Method Definitions

18.7.1 Function _buildPadawanState

- OK

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

18.7.2 Function _calculateVotes

- OK

```
    function _calculateVotes(
        uint128 yes,
        uint128 no
    ) private view returns (bool) {
        bool passed = false;
        passed = _softMajority(yes, no);
        return passed;
    }
```

18.7.3 Function _changeState

- OK

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

18.7.4 Function _finalize

- OK

```
    function _finalize(bool passed) private {
        _results = ProposalResults(
            uint32(0),
            passed,
            _proposalInfo.votesFor,
            _proposalInfo.votesAgainst,
            _voteTotal,
            _voteCountModel,
            uint32(now)
        );

        ProposalState state = passed ? ProposalState.Passed : ProposalState.NotPassed;

        _changeState(state);

        IClient(address(_client)).onProposalPassed{value: 1 ton} (_proposalInfo);
    }
```

18.7.5 Function _findInWhiteList

- OK

```
    function _findInWhiteList(address padawanOwner) view private returns (bool) {
        for(uint32 index = 0; index < _whiteList.length; index++) {
            if(_whiteList[index] == padawanOwner) {
                return true;
            }
        }
        return false;
    }
```

18.7.6 Function _getGroupMembers

- OK

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

18.7.7 Function _softMajority

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

```
    function _softMajority(
        uint128 yes,
        uint128 no
    ) private view returns (bool) {
        bool passed = false;
        passed = yes >= 1 + (_voteTotal / 10) + (no * ((_voteTotal
            / 2) - (_voteTotal / 10))) / (_voteTotal / 2);
        return passed;
    }
```

18.7.8 Function _tryEarlyComplete

- **Major issue: Overflow in `Proposal._tryEarlyComplete`**
  If vote counts are expected to be in the full `uint32` range, `yes*2` and `no*2` can overflow. Fix: use `uint64` for parameters.
- Minor issue (readability): use `returns (bool completed, bool passed)` to avoid the need to define temporary variables and to return them.

```
    function _tryEarlyComplete(
        uint128 yes,
        uint128 no
    ) private view returns (bool, bool) {
        (bool completed, bool passed) = (false, false);
        if (yes * 2 > _voteTotal) {
            completed = true;
            passed = true;
        } else if(no * 2 >= _voteTotal) {
            completed = true;
            passed = false;
        }
        return (completed, passed);
    }
```

18.7.9 Function _wrapUp

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

```
    function _wrapUp() private {
        (bool completed, bool passed) = (false, false);

        if (now > _proposalInfo.end) {
            completed = true;
            passed = _calculateVotes(_proposalInfo.votesFor,
                _proposalInfo.votesAgainst);
        } else {
            (completed, passed) = _tryEarlyComplete(_proposalInfo.
                votesFor, _proposalInfo.votesAgainst);
        }

        if (completed) {
            _changeState(ProposalState.Ended);
            _finalize(passed);
        }
    }
```

Chapter 19

Contract ProposalFactory

Contents

19.1 Overview

In file `ProposalFactory.sol`

19.2 Contract Inheritance

Base	

19.3 Static Variable Definitions

- OK

```
    address static _deployer;
```

19.4 Constructor Definitions

19.4.1 Constructor

- OK

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

19.5 Public Method Definitions

19.5.1 Function deployAddMemberProposal

- OK

```
    function deployAddMemberProposal(
        address client,
        string title,
        uint128 votePrice,
        uint128 voteTotal,
        address voteProvider,
        address group,
        address[] whiteList,
        AddMemberProposalSpecific specific
    ) external view onlyContract {
        require(msg.value >= DEPLOY_PROPOSAL_PAY + 1 ton);
        TvmBuilder b;
        b.store(specific);
        TvmCell cellSpecific = b.toCell();
        ISmvRoot(_deployer).deployProposal
            {value: 0, flag: 64, bounce: true}
            (
                client,
                title,
                votePrice,
                voteTotal,
                voteProvider,
                group,
                whiteList,
                'add-member',
                cellSpecific
            );
    }
```

19.5.2 Function deployContestProposal

- OK

```
    function deployContestProposal(
        address client,
        string title,
        address group,
        ContestProposalSpecific specific
    ) external view onlyContract {
        require(msg.value >= DEPLOY_PROPOSAL_PAY + 1 ton);
        TvmBuilder b;
        b.store(specific);
        TvmCell cellSpecific = b.toCell();
        address[] arr;
        ISmvRoot(_deployer).deployProposal
            {value: 0, flag: 64, bounce: true}
            (
                client,
                title,
                1 ton,
                1000000000,
                address(0),
                group,
                arr,
                'contest',
                cellSpecific
            );
    }
```

19.5.3 Function deployRemoveMemberProposal

- OK

```
    function deployRemoveMemberProposal(
        address client,
        string title,
        uint128 votePrice,
        uint128 voteTotal,
        address voteProvider,
        address group,
        address[] whiteList,
        RemoveMemberProposalSpecific specific
    ) external view onlyContract {
        require(msg.value >= DEPLOY_PROPOSAL_PAY + 1 ton);
        TvmBuilder b;
        b.store(specific);
        TvmCell cellSpecific = b.toCell();
        ISmvRoot(_deployer).deployProposal
            {value: 0, flag: 64, bounce: true}
            (
                client,
                title,
                votePrice,
                voteTotal,
                voteProvider,
                group,
                whiteList,
                'remove-member',
                cellSpecific
            );
    }
```

Chapter 20

Contract ProposalFactoryResolver

Contents

20.1 Overview

In file `ProposalFactoryResolver.sol`

20.2 Variable Definitions

```
    TvmCell _codeProposalFactory;
```

20.3 Public Method Definitions

20.3.1 Function resolveProposalFactory

- OK

```
    function resolveProposalFactory(address deployer) public view
        returns (address addrProposalFactory) {
        TvmCell state = _buildProposalFactoryState(deployer);
        uint256 hashState = tvm.hash(state);
        addrProposalFactory = address.makeAddrStd(0, hashState);
    }
```

20.4 Internal Method Definitions

20.4.1 Function _buildProposalFactoryState

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

```
    function _buildProposalFactoryState(address deployer) internal
        view returns (TvmCell) {
        return tvm.buildStateInit({
            contr: ProposalFactory,
            varInit: {_deployer: deployer},
            code: _codeProposalFactory
        });
    }
```

Chapter 21

Contract ProposalResolver

Contents

21.1 Overview

In file `ProposalResolver.sol`

21.2 Variable Definitions

```
    TvmCell _codeProposal;
```

21.3 Public Method Definitions

21.3.1 Function resolveProposal

- OK

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

21.4 Internal Method Definitions

21.4.1 Function _buildProposalState

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

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

Chapter 22

Contract SmvRoot

Contents

22.1 Overview

In file `SmvRoot.sol`

22.2 Contract Inheritance

Base	
ISmvRoot	
ISmvRootStoreCallback	
PadawanResolver	
ProposalResolver	
GroupResolver	
ProposalFactoryResolver	
Checks	

22.3 Constant Definitions

```
    uint8 constant CHECK_PROPOSAL = 1;
```

```
    uint8 constant CHECK_PADAWAN = 2;
```

```
    uint8 constant CHECK_GROUP = 4;
```

```
    uint8 constant CHECK_PROPOSAL_FACTORY = 8;
```

```
    uint8 constant CHECK_BFTG_ROOT_ADDRESS = 16;
```

22.4 Variable Definitions

```
    address _addrSmvRootStore;
```

```
    address _addrBftgRoot;
```

```
    address _addrProposalFactory;
```

```
    bool public _inited = false;
```

```
    uint32 _deployedPadawansCounter;
```

```
    uint32 _deployedProposalsCounter;
```

22.5 Modifier Definitions

22.5.1 Modifier onlyStore

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

22.6 Constructor Definitions

22.6.1 Constructor

Critical issue: Constructor for SmvRoot (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    constructor(address addrSmvRootStore) public {
        if (msg.sender == address(0)) {
            require(msg.pubkey() == tvm.pubkey(), Errors.
                ONLY_SIGNED);
        }
        require(addrSmvRootStore != address(0), Errors.
            STORE_UNDEFINED);
        tvm.accept();

        _addrSmvRootStore = addrSmvRootStore;
        ISmvRootStore(_addrSmvRootStore).queryCode
            {value: 0.2 ton, bounce: true}
            (ContractCode.Proposal);
        ISmvRootStore(_addrSmvRootStore).queryCode
            {value: 0.2 ton, bounce: true}
            (ContractCode.Padawan);
        ISmvRootStore(_addrSmvRootStore).queryCode
            {value: 0.2 ton, bounce: true}
            (ContractCode.Group);
        ISmvRootStore(_addrSmvRootStore).queryCode
            {value: 0.2 ton, bounce: true}
            (ContractCode.ProposalFactory);
        ISmvRootStore(_addrSmvRootStore).queryAddr
            {value: 0.2 ton, bounce: true}
            (ContractAddr.BftgRoot);

        _createChecks();
    }
```

22.7 Public Method Definitions

22.7.1 Function _deployProposal

- TODO

```
    function _deployProposal(
        address client,
        string title,
        uint128 votePrice,
        uint128 voteTotal,
        address voteProvider,
        address group,
        address[] whiteList,
        string proposalType,
        TvmCell specific
    ) public onlyMe {
        TvmCell state = _buildProposalState(
            _deployedProposalsCounter);
        new Proposal {stateInit: state, value: START_BALANCE}(
            client,
            title,
            votePrice,
            voteTotal,
            voteProvider,
            group,
            whiteList,
            proposalType,
            specific,
            _codePadawan
        );
        _deployedProposalsCounter++;
    }
```

22.7.2 Function deployGroup

- TODO

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

22.7.3 Function deployPadawan

- TODO

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

22.7.4 Function deployProposal

- TODO

```
    function deployProposal(
        address client,
        string title,
        uint128 votePrice,
        uint128 voteTotal,
        address voteProvider,
        address group,
        address[] whiteList,
        string proposalType,
        TvmCell specific
    ) external override onlyContract {
        require(msg.sender == _addrProposalFactory);
        require(msg.value >= DEPLOY_PROPOSAL_FEE);
        TvmBuilder b;
        b.store(specific);
        TvmCell cellSpecific = b.toCell();
        _beforeProposalDeploy(
            client,
            title,
            votePrice,
            voteTotal,
            voteProvider,
            group,
            whiteList,
            proposalType,
            cellSpecific
        );
    }
```

22.7.5 Function getStats

- TODO

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

22.7.6 Function getStored

- TODO

```
    function getStored() public view returns (
        TvmCell codePadawan,
        TvmCell codeProposal,
        TvmCell codeGroup,
        TvmCell codeProposalFactory,
        address addrBftgRoot,
        address proposalFactory
    ) {
        codePadawan = _codePadawan;
        codeProposal = _codeProposal;
        codeGroup = _codeGroup;
        codeProposalFactory = _codeProposalFactory;
        addrBftgRoot = _addrBftgRoot;
        proposalFactory = _addrProposalFactory;
    }
```

22.7.7 Function updateAddr

- TODO

```
    function updateAddr(ContractAddr kind, address addr) external
        override onlyStore {
        require(addr != address(0));
        if (kind == ContractAddr.BftgRoot) {
            _addrBftgRoot = addr;
            _passCheck(CHECK_BFTG_ROOT_ADDRESS);
        }
        _onInit();
    }
```

22.7.8 Function updateCode

- TODO

```
    function updateCode(
        ContractCode kind,
        TvmCell code
    ) external override onlyStore {
        if (kind == ContractCode.Proposal) {
            _codeProposal = code;
            _passCheck(CHECK_PROPOSAL);
        } else if (kind == ContractCode.Padawan) {
            _codePadawan = code;
            _passCheck(CHECK_PADAWAN);
        } else if (kind == ContractCode.Group) {
            _codeGroup = code;
            _passCheck(CHECK_GROUP);
        } else if (kind == ContractCode.ProposalFactory) {
            _codeProposalFactory = code;
            _passCheck(CHECK_PROPOSAL_FACTORY);
        }
        _onInit();
    }
```

22.8 Internal Method Definitions

22.8.1 Function _beforeProposalDeploy

- TODO

```
    function _beforeProposalDeploy(
        address client,
        string title,
        uint128 votePrice,
        uint128 voteTotal,
        address voteProvider,
        address group,
        address[] whiteList,
        string proposalType,
        TvmCell specific
    ) private view {
        TvmCell state = _buildProposalState(
            _deployedProposalsCounter);
        uint256 hashState = tvm.hash(state);
        address proposal = address.makeAddrStd(0, hashState);
        // IClient(_addrDensRoot).onProposalDeploy
        //     {value: 1 ton, bounce: true}
        //     (proposal, proposalType, specific);
        this._deployProposal
            {value: 4 ton}
            (client, title, votePrice, voteTotal, voteProvider,
                group, whiteList, proposalType, specific);
    }
```

22.8.2 Function _createChecks

- TODO

```
    function _createChecks() private inline {
        _checkList =
            CHECK_PROPOSAL |
            CHECK_PADAWAN |
            CHECK_GROUP |
            CHECK_PROPOSAL_FACTORY |
            CHECK_BFTG_ROOT_ADDRESS;
    }
```

22.8.3 Function _onInit

- TODO

```
    function _onInit() private {
        if(_isCheckListEmpty() && !_inited) {
            _inited = true;
            TvmCell state = _buildProposalFactoryState(address(this
                ));
            _addrProposalFactory = new ProposalFactory
                {stateInit: state, value: 0.2 ton}
                ();
        }
    }
```

Chapter 23

Contract SmvRootStore

Contents

23.1 Overview

In file `SmvRootStore.sol`

23.2 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).

23.3 Contract Inheritance

Base	
ISmvRootStore	

23.4 Variable Definitions

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

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

23.5 Public Method Definitions

23.5.1 Function queryAddr

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

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

23.5.2 Function queryCode

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

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

23.5.3 Function setBftgRootAddr

- OK

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

23.5.4 Function setGroupCode

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

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

23.5.5 Function setPadawanCode

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

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

23.5.6 Function setProposalCode

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

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

23.5.7 Function setProposalFactoryCode

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

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