

Audit of the BFTG project

By OCamlPro

August 20, 2021

Table of Major and Critical Issues

Critical issue: Administrative Take-over in <code>BftgRoot.constructor</code>	49
Major issue: No initialization check performed in <code>BftgRoot.constructor</code>	50
Critical issue: <code>tvm.accept</code> without check in <code>BftgRoot.deployContest</code>	51
Major issue: Non-reentrant in <code>BftgRoot.registerMemberJuryGroup</code>	52
Critical issue: No stage check in <code>Contest.calcRewards</code>	62
Major issue: Wrong computation in <code>Contest.calcRewards</code>	62
Critical issue: Missing permission checks in <code>Contest.changeStage</code>	62
Critical issue: Multiple revelations in <code>Contest.reveal</code>	64
Major issue: Unbounded storage in <code>Contest.submit</code>	65
Critical issue: No permission check in <code>Contest.updateCode</code>	65
Major issue: No gas check in <code>Contest.updateCode</code>	65
Critical issue: No permission check in <code>Group.constructor</code>	71
Critical issue: No permission check in <code>Group.addMember</code>	71
Critical issue: No permission check on <code>removeMember</code>	72
Major issue: Wrong comparison in <code>JuryGroup.withdraw</code>	77
Major issue: Incorrect computation in <code>Padawan.onEstimateVotes</code>	83
Critical issue: Can empty voting rights in <code>Padawan.onTokenWalletDeploy</code>	84
Critical issue: Unbounded voting rights in <code>Padawan.onTokenWalletGetBalance</code>	85
Critical issue: Race condition in <code>Padawan.reclaimDeposit</code>	85
Critical issue: No permission check on <code>Proposal.onGetMembers</code>	95
Critical issue: Division by 0 in <code>Proposal._softMajority</code>	98
Major issue: Overflow in <code>Proposal._tryEarlyComplete</code>	98
Critical issue: Constructor for <code>SmvRoot</code> (fake)	110

Contents

1	Introduction	11
1.0.1	Location	11
1.0.2	End Date	11
2	Overview	12
3	Library Modules	13
3.1	Module "BFTG.sol"	14
3.1.1	Imports	14
3.2	Module "Errors.sol"	15
3.2.1	Pragmas	15
3.2.2	Contract Definitions	15
3.3	Module "Glossary.sol"	16
3.3.1	Pragmas	16
3.3.2	Type Definitions	16
3.3.2.1	Enum VoteCountModel	16
3.3.2.2	Enum ProposalType	16
3.3.2.3	Enum ProposalState	17
3.4	Module "IContest.sol"	18
3.4.1	Pragmas	18
3.4.2	Type Definitions	18
3.4.2.1	Enum ContestStage	18
3.4.2.2	Struct Submission	18
3.4.2.3	Struct HiddenVote	19
3.4.2.4	Struct RevealVote	19
3.4.2.5	Struct Vote	19
3.4.2.6	Struct Reward	19
4	Interface Modules	20
4.1	Module "IBftgRoot.sol"	21
4.1.1	Pragmas	21
4.1.2	Type Definitions	21
4.1.2.1	Struct JuryGroupPending	21
4.1.3	Contract Definitions	21

4.2	Module "IBftgRootStore.sol"	22
4.2.1	Pragmas	22
4.2.2	Type Definitions	22
4.2.2.1	Enum ContractCode	22
4.2.2.2	Enum ContractAddr	22
4.2.3	Contract Definitions	22
4.3	Module "IClient.sol"	23
4.3.1	Pragmas	23
4.3.2	Imports	23
4.3.3	Contract Definitions	23
4.4	Module "IGroup.sol"	24
4.4.1	Pragmas	24
4.4.2	Contract Definitions	24
4.5	Module "IJuryGroup.sol"	25
4.5.1	Pragmas	25
4.5.2	Type Definitions	25
4.5.2.1	Struct Member	25
4.5.3	Contract Definitions	25
4.6	Module "IPadawan.sol"	26
4.6.1	Pragmas	26
4.6.2	Imports	26
4.6.3	Type Definitions	26
4.6.3.1	Struct TipAccount	26
4.6.4	Contract Definitions	26
4.7	Module "IProposal.sol"	27
4.7.1	Pragmas	27
4.7.2	Imports	27
4.7.3	Type Definitions	27
4.7.3.1	Struct ProposalResults	27
4.7.3.2	Struct ProposalInfo	27
4.7.4	Contract Definitions	28
4.8	Module "ITokenRoot.sol"	29
4.8.1	Pragmas	29
4.8.2	Contract Definitions	29
4.9	Module "ITokenWallet.sol"	30
4.9.1	Pragmas	30
4.9.2	Contract Definitions	30
5	Contract Modules	31
5.1	Module "Base.sol"	32
5.1.1	Pragmas	32
5.1.2	Imports	32
5.1.3	Contract Definitions	32
5.2	Module "BftgRoot.sol"	33
5.2.1	Pragmas	33
5.2.2	Imports	33

5.2.3	Contract Definitions	33
5.3	Module "Checks.sol"	34
5.3.1	Pragmas	34
5.3.2	Contract Definitions	34
5.4	Module "Contest.sol"	35
5.4.1	Pragmas	35
5.4.2	Imports	35
5.4.3	Contract Definitions	35
5.5	Module "ContestResolver.sol"	36
5.5.1	Pragmas	36
5.5.2	Imports	36
5.5.3	Contract Definitions	36
5.6	Module "Group.sol"	37
5.6.1	Pragmas	37
5.6.2	Imports	37
5.6.3	Contract Definitions	37
5.7	Module "GroupResolver.sol"	38
5.7.1	Pragmas	38
5.7.2	Imports	38
5.7.3	Contract Definitions	38
5.8	Module "JuryGroup.sol"	39
5.8.1	Pragmas	39
5.8.2	Imports	39
5.8.3	Contract Definitions	39
5.9	Module "JuryGroupResolver.sol"	40
5.9.1	Pragmas	40
5.9.2	Imports	40
5.9.3	Contract Definitions	40
5.10	Module "Padawan.sol"	41
5.10.1	Pragmas	41
5.10.2	Imports	41
5.10.3	Type Definitions	41
5.10.3.1	Struct PadawanData	41
5.10.3.2	Struct Balance	41
5.10.3.3	Struct ActiveProposal	41
5.10.3.4	Struct Reclaim	42
5.10.4	Contract Definitions	42
5.11	Module "PadawanResolver.sol"	43
5.11.1	Pragmas	43
5.11.2	Imports	43
5.11.3	Contract Definitions	43
5.12	Module "Proposal.sol"	44
5.12.1	Pragmas	44
5.12.2	Imports	44
5.12.3	Contract Definitions	44

6	Contract Base	45
6.1	Overview	45
6.2	Constant Definitions	45
6.3	Modifier Definitions	46
6.3.1	Modifier signed	46
6.3.2	Modifier accept	46
6.3.3	Modifier onlyContract	47
6.3.4	Modifier onlyMe	47
7	Contract BftgRoot	48
7.1	Overview	48
7.2	Contract Inheritance	49
7.3	Constant Definitions	49
7.4	Variable Definitions	49
7.5	Modifier Definitions	49
7.5.1	Modifier onlyStore	49
7.6	Constructor Definitions	49
7.6.1	Constructor	49
7.7	Public Method Definitions	50
7.7.1	OnBounce function	50
7.7.2	Function deployContest	51
7.7.3	Function deployJuryGroup	51
7.7.4	Function getMembersCallback	51
7.7.5	Function getStored	52
7.7.6	Function registerMemberJuryGroup	52
7.7.7	Function updateAddr	52
7.7.8	Function updateCode	53
7.8	Internal Method Definitions	53
7.8.1	Function _createChecks	53
7.8.2	Function _onInit	53
8	Contract BftgRootStore	54
8.1	Overview	54
8.2	General Minor-level Remarks	54
8.3	Contract Inheritance	55
8.4	Variable Definitions	55
8.5	Public Method Definitions	55
8.5.1	Function queryAddr	55
8.5.2	Function queryCode	56
8.5.3	Function setContestCode	56
8.5.4	Function setJuryGroupCode	56

9	Contract Checks	57
9.1	Overview	57
9.2	Variable Definitions	57
9.3	Modifier Definitions	57
9.3.1	Modifier checksEmpty	57
9.4	Internal Method Definitions	58
9.4.1	Function _isCheckListEmpty	58
9.4.2	Function _passCheck	58
10	Contract Contest	59
10.1	Overview	60
10.2	Contract Inheritance	60
10.3	Constant Definitions	60
10.4	Static Variable Definitions	60
10.5	Variable Definitions	60
10.6	Constructor Definitions	61
10.6.1	Constructor	61
10.7	Public Method Definitions	61
10.7.1	OnBounce function	61
10.7.2	Function calcRewards	62
10.7.3	Function changeStage	62
10.7.4	Function claimPartisipantReward	62
10.7.5	Function getHiddenVotesByAddress	63
10.7.6	Function getMembersCallback	63
10.7.7	Function hashVote	63
10.7.8	Function reveal	64
10.7.9	Function stakePartisipantReward	64
10.7.10	Function submit	65
10.7.11	Function updateAddr	65
10.7.12	Function updateCode	65
10.7.13	Function vote	66
10.8	Internal Method Definitions	66
10.8.1	Function _calcPointValue	66
10.8.2	Function _changeStage	66
10.8.3	Function _createChecks	67
10.8.4	Function _onInit	67
11	Contract ContestResolver	68
11.1	Overview	68
11.2	Variable Definitions	68
11.3	Public Method Definitions	68
11.3.1	Function resolveContest	68
11.4	Internal Method Definitions	69
11.4.1	Function _buildContestState	69

12 Contract Group	70
12.1 Overview	70
12.2 Contract Inheritance	70
12.3 Static Variable Definitions	70
12.4 Variable Definitions	71
12.5 Constructor Definitions	71
12.5.1 Constructor	71
12.6 Public Method Definitions	71
12.6.1 Function addMember	71
12.6.2 Function getMembers	71
12.6.3 Function removeMember	72
13 Contract GroupResolver	73
13.1 Overview	73
13.2 Variable Definitions	73
13.3 Public Method Definitions	73
13.3.1 Function resolveGroup	73
13.4 Internal Method Definitions	74
13.4.1 Function _buildGroupState	74
14 Contract JuryGroup	75
14.1 Overview	75
14.2 Contract Inheritance	75
14.3 Static Variable Definitions	75
14.4 Variable Definitions	76
14.5 Modifier Definitions	76
14.5.1 Modifier onlyDeployer	76
14.6 Constructor Definitions	76
14.6.1 Constructor	76
14.7 Public Method Definitions	76
14.7.1 Function getMembers	76
14.7.2 Function registerMember	77
14.7.3 Function withdraw	77
14.8 Internal Method Definitions	77
14.8.1 Function _addMember	77
15 Contract JuryGroupResolver	78
15.1 Overview	78
15.2 Variable Definitions	78
15.3 Public Method Definitions	78
15.3.1 Function resolveJuryGroup	78
15.4 Internal Method Definitions	79
15.4.1 Function _buildJuryGroupState	79

16 Contract Padawan	80
16.1 Overview	80
16.2 Contract Inheritance	81
16.3 Static Variable Definitions	81
16.4 Variable Definitions	81
16.5 Modifier Definitions	81
16.5.1 Modifier onlyOwner	81
16.6 Constructor Definitions	82
16.6.1 Constructor	82
16.7 Public Method Definitions	82
16.7.1 Function confirmVote	82
16.7.2 Function createTokenAccount	82
16.7.3 Function depositTokens	83
16.7.4 Function depositTons	83
16.7.5 Function onEstimateVotes	83
16.7.6 Function onTokenWalletDeploy	84
16.7.7 Function onTokenWalletGetBalance	85
16.7.8 Function reclaimDeposit	85
16.7.9 Function rejectVote	86
16.7.10 Function updateStatus	86
16.7.11 Function vote	87
16.8 Internal Method Definitions	88
16.8.1 Function _doReclaim	88
17 Contract PadawanResolver	89
17.1 Overview	89
17.2 Variable Definitions	89
17.3 Public Method Definitions	89
17.3.1 Function resolvePadawan	89
17.4 Internal Method Definitions	90
17.4.1 Function _buildPadawanState	90
18 Contract Proposal	91
18.1 Overview	92
18.2 Contract Inheritance	92
18.3 Static Variable Definitions	92
18.4 Variable Definitions	92
18.5 Constructor Definitions	93
18.5.1 Constructor	93
18.6 Public Method Definitions	94
18.6.1 Function estimateVotes	94
18.6.2 Function getAll	94
18.6.3 Function getCurrentVotes	94
18.6.4 Function getInfo	94
18.6.5 Function getVotingResults	94
18.6.6 Function onGetMembers	95

18.6.7	Function queryStatus	95
18.6.8	Function vote	95
18.6.9	Function wrapUp	96
18.7	Internal Method Definitions	96
18.7.1	Function _buildPadawanState	96
18.7.2	Function _calculateVotes	96
18.7.3	Function _changeState	97
18.7.4	Function _finalize	97
18.7.5	Function _findInWhiteList	97
18.7.6	Function _getGroupMembers	98
18.7.7	Function _softMajority	98
18.7.8	Function _tryEarlyComplete	98
18.7.9	Function _wrapUp	99
19	Contract ProposalFactory	100
19.1	Overview	100
19.2	Contract Inheritance	100
19.3	Static Variable Definitions	100
19.4	Constructor Definitions	101
19.4.1	Constructor	101
19.5	Public Method Definitions	101
19.5.1	Function deployAddMemberProposal	101
19.5.2	Function deployContestProposal	101
19.5.3	Function deployRemoveMemberProposal	102
20	Contract ProposalFactoryResolver	104
20.1	Overview	104
20.2	Variable Definitions	104
20.3	Public Method Definitions	104
20.3.1	Function resolveProposalFactory	104
20.4	Internal Method Definitions	105
20.4.1	Function _buildProposalFactoryState	105
21	Contract ProposalResolver	106
21.1	Overview	106
21.2	Variable Definitions	106
21.3	Public Method Definitions	106
21.3.1	Function resolveProposal	106
21.4	Internal Method Definitions	107
21.4.1	Function _buildProposalState	107
22	Contract SmvRoot	108
22.1	Overview	108
22.2	Contract Inheritance	109
22.3	Constant Definitions	109
22.4	Variable Definitions	109

22.5	Modifier Definitions	109
22.5.1	Modifier onlyStore	109
22.6	Constructor Definitions	110
22.6.1	Constructor	110
22.7	Public Method Definitions	110
22.7.1	Function _deployProposal	110
22.7.2	Function deployGroup	111
22.7.3	Function deployPadawan	111
22.7.4	Function deployProposal	111
22.7.5	Function getStats	112
22.7.6	Function getStored	112
22.7.7	Function updateAddr	113
22.7.8	Function updateCode	113
22.8	Internal Method Definitions	113
22.8.1	Function _beforeProposalDeploy	113
22.8.2	Function _createChecks	114
22.8.3	Function _onInit	114
23	Contract SmvRootStore	115
23.1	Overview	115
23.2	General Minor-level Remarks	115
23.3	Contract Inheritance	116
23.4	Variable Definitions	116
23.5	Public Method Definitions	116
23.5.1	Function queryAddr	116
23.5.2	Function queryCode	116
23.5.3	Function setBftgRootAddr	117
23.5.4	Function setGroupCode	117
23.5.5	Function setPadawanCode	117
23.5.6	Function setProposalCode	117
23.5.7	Function setProposalFactoryCode	117

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 Pragas

ton	-solidity >=0.37.0	
-----	--------------------	--

3.2.2 Contract Definitions

- Errors

3.3 Module "Glossary.sol"

3.3.1 Pragas

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 Pragas

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 Pragas

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 Pragas

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 Pragas

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 Pragas

ton	-solidity >= 0.36.0	
-----	---------------------	--

4.4.2 Contract Definitions

- IGroup
- IGroupCallback

4.5 Module "IJuryGroup.sol"

4.5.1 Pragas

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 Pragas

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 Pragas

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 Pragas

ton	-solidity >= 0.42.0	
-----	---------------------	--

4.8.2 Contract Definitions

- ITokenRoot

4.9 Module "ITokenWallet.sol"

4.9.1 Pragas

ton	-solidity >= 0.42.0	
-----	---------------------	--

4.9.2 Contract Definitions

- ITokenWallet

Chapter 5

Contract Modules

5.1 Module "Base.sol"

5.1.1 Pragas

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 Pragas

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 Pragas

ton	-solidity >= 0.42.0	
-----	---------------------	--

5.3.2 Contract Definitions

- Checks

5.4 Module "Contest.sol"

5.4.1 Pragas

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 Pragas

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 Pragas

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 Pragas

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 Pragas

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 Pragas

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 Pragma

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 Pragas

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 Pragas

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	45
6.2	Constant Definitions	45
6.3	Modifier Definitions	46
6.3.1	Modifier signed	46
6.3.2	Modifier accept	46
6.3.3	Modifier onlyContract	47
6.3.4	Modifier onlyMe	47

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() == tvn.pubkey(), 100);
34      tvn.accept();
35      -;
36  }

```

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.

```

38  modifier accept {
39      tvn.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         -;  
52     }
```


Chapter 7

Contract BftgRoot

Contents

7.1	Overview	48
7.2	Contract Inheritance	49
7.3	Constant Definitions	49
7.4	Variable Definitions	49
7.5	Modifier Definitions	49
7.5.1	Modifier onlyStore	49
7.6	Constructor Definitions	49
7.6.1	Constructor	49
7.7	Public Method Definitions	50
7.7.1	OnBounce function	50
7.7.2	Function deployContest	51
7.7.3	Function deployJuryGroup	51
7.7.4	Function getMembersCallback	51
7.7.5	Function getStored	52
7.7.6	Function registerMemberJuryGroup	52
7.7.7	Function updateAddr	52
7.7.8	Function updateCode	53
7.8	Internal Method Definitions	53
7.8.1	Function _createChecks	53
7.8.2	Function _onInit	53

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() == tvn.pubkey(), Errors.
                ONLY_SIGNED);
39         }
40         require(addrBftgRootStore != address(0), Errors.
            STORE_UNDEFINED);
41         tvn.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
121     ) public override {
122         address addrContest = resolveContest(address(this));
123         address addrJuryGroup = resolveJuryGroup(tag, address(this));
124         require(msg.sender == addrContest || address(this) == msg.
125             sender, 105);
126         _juryGroupPendings[addrJuryGroup] = JuryGroupPending(
127             addrMember, tag);
128         IJuryGroup(addrJuryGroup).getMembers
129             {value: 0, bounce: true, flag: 64}
130             ();
131     }

```

7.7.7 Function updateAddr

- OK

```

77     function updateAddr(ContractAddr kind, address addr) external
78         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 BftgRootStore

Contents

8.1	Overview	54
8.2	General Minor-level Remarks	54
8.3	Contract Inheritance	55
8.4	Variable Definitions	55
8.5	Public Method Definitions	55
8.5.1	Function queryAddr	55
8.5.2	Function queryCode	56
8.5.3	Function setContestCode	56
8.5.4	Function setJuryGroupCode	56

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

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

```
11 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.

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


8.5.2 Function queryCode

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

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

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

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`

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

Chapter 9

Contract Checks

Contents

9.1 Overview	57
9.2 Variable Definitions	57
9.3 Modifier Definitions	57
9.3.1 Modifier checksEmpty	57
9.4 Internal Method Definitions	58
9.4.1 Function _isCheckListEmpty	58
9.4.2 Function _passCheck	58

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

```
4 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`.

```
12     modifier checksEmpty() {
13         require(_isCheckListEmpty(), 100); //Errors.
14             NOT_ALL_CHECKS_PASSED);
15         tvm.accept();
16         -;
17     }
```

9.4 Internal Method Definitions

9.4.1 Function `_isCheckListEmpty`

- OK

```
9     function _isCheckListEmpty() internal view inline returns (bool
10         ) {
11         return (_checkList == 0);
12     }
```

9.4.2 Function `_passCheck`

- OK

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

Chapter 10

Contract Contest

Contents

10.1 Overview	60
10.2 Contract Inheritance	60
10.3 Constant Definitions	60
10.4 Static Variable Definitions	60
10.5 Variable Definitions	60
10.6 Constructor Definitions	61
10.6.1 Constructor	61
10.7 Public Method Definitions	61
10.7.1 OnBounce function	61
10.7.2 Function calcRewards	62
10.7.3 Function changeStage	62
10.7.4 Function claimPartisipantReward	62
10.7.5 Function getHiddenVotesByAddress	63
10.7.6 Function getMembersCallback	63
10.7.7 Function hashVote	63
10.7.8 Function reveal	64
10.7.9 Function stakePartisipantReward	64
10.7.10 Function submit	65
10.7.11 Function updateAddr	65
10.7.12 Function updateCode	65
10.7.13 Function vote	66
10.8 Internal Method Definitions	66
10.8.1 Function _calcPointValue	66
10.8.2 Function _changeStage	66
10.8.3 Function _createChecks	67
10.8.4 Function _onInit	67

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

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

10.4 Static Variable Definitions

- OK

```
28  address static _deployer;
```

10.5 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;

```

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.

```

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 }

```

10.7 Public Method Definitions

10.7.1 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 }

```

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.

```

175 function calcRewards() public {
176     _calcPointValue();
177     optional(uint32, Vote[]) optSubmissionVotes =
178         _submissionVotes.min();
179     while (optSubmissionVotes.hasValue()) {
180         (uint32 id, Vote[] submissionVotes) =
181             optSubmissionVotes.get();
182         for(uint8 i = 0; i < submissionVotes.length; i++) {
183             _rewards[_submissions[id].addrPartisipant].total +=
184                 submissionVotes[i].score * _pointValue;
185         }
186         optSubmissionVotes = _submissionVotes.next(id);
187     }
188     _changeStage(ContestStage.Reward);
189 }

```

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`.

```

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

```

10.7.4 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     }

```

10.7.5 Function getHiddenVotesByAddress

- OK

```

145     function getHiddenVotesByAddress(address juryAddr) public view
146         returns (mapping(uint32 => HiddenVote) hiddenVotes) {
147         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`.

```

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

```

10.7.7 Function hashVote

- OK

```

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

```

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
160                 [i].submissionId].hash;
161             uint newHash = hashVote(revealVotes[i].submissionId,
162                 revealVotes[i].score, revealVotes[i].comment);
163             require(oldHash == newHash, 106);
164             _submissionVotes[revealVotes[i].submissionId].push(Vote
165                 (msg.sender, revealVotes[i].score, revealVotes[i].
166                 comment));
167         }
168         msg.sender.transfer(0, true, 64);
169     }

```

10.7.9 Function stakePartisipantReward

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

```

204     function stakePartisipantReward(uint128 amount, string tag,
205         address addrJury) public {
206         require(_rewards.exists(msg.sender), 107);
207         require(_rewards[msg.sender].total - _rewards[msg.sender].
208             paid >= amount, 108);
209         bool isTagExists = false;
210         for(uint8 i = 0; i < _tags.length; i++) {
211             if(_tags[i] == tag) isTagExists = true;
212         }
213         require(isTagExists, 108);
214         _rewards[msg.sender].paid += amount;
215         IBftgRoot(_deployer).registerMemberJuryGroup
216             {value: amount, bounce: true, flag: 2}
217             (tag, addrJury == address(0) ? msg.sender : addrJury);
218         msg.sender.transfer(0, true, 64);
219     }

```

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.

```

121 function submit(address addrPartisipant, string forumLink,
122               string fileLink, uint hash) external {
123     require(_stage == ContestStage.Underway, 104);
124     _submissions[_submissionsCounter] = (Submission(
125       _submissionsCounter, addrPartisipant, forumLink,
126       fileLink, hash, uint32(now)));
127     _submissionsCounter += 1;
128     msg.sender.transfer(0, true, 64);
129 }

```

10.7.11 Function updateAddr

- OK

```

81 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`.

```

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     }

```

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

```

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].
139                 submissionId)) {
140                 _juryHiddenVotes[msg.sender][hiddenVotes[i].
141                     submissionId] = hiddenVotes[i];
142             }
143         }
144         msg.sender.transfer(0, true, 64);
145     }

```

10.8 Internal Method Definitions

10.8.1 Function _calcPointValue

- OK

```

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

```

10.8.2 Function _changeStage

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

```

106     function _changeStage(ContestStage stage) private inline
107         returns (ContestStage) {
108         require(_stage < stage, 103);
109         if (stage == ContestStage.Underway) {
110             _underwayEnds = uint32(now) + _underwayDuration;
111         }
112         _stage = stage;
113     }

```

10.8.3 Function _createChecks

- OK

```

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

```

10.8.4 Function _onInit

- TODO

```

47     function _onInit() private {
48         if(!_isCheckListEmpty() && !_initiated) {
49             _initiated = true;
50             for(uint8 i = 0; i < _tags.length; i++) {
51                 TvmCell state = _buildJuryGroupState(_tags[i],
52                 _deployer);
53                 uint256 hashState = tvm.hash(state);
54                 address addrJuryGroup = address.makeAddrStd(0,
55                 hashState);
56                 _tagsPendings[addrJuryGroup] = true;
57                 IJuryGroup(addrJuryGroup).getMembers{
58                     value: 0.2 ton,
59                     flag: 1,
60                     bounce: true
61                 }();
62             }
63         }
64     }

```

Chapter 11

Contract ContestResolver

Contents

11.1 Overview	68
11.2 Variable Definitions	68
11.3 Public Method Definitions	68
11.3.1 Function resolveContest	68
11.4 Internal Method Definitions	69
11.4.1 Function _buildContestState	69

11.1 Overview

In file ContestResolver.sol

11.2 Variable Definitions

```
8     TvmCell _codeContest;
```

11.3 Public Method Definitions

11.3.1 Function resolveContest

- OK

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

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.

```
16     function _buildContestState(address deployer) internal virtual
17         view returns (TvmCell) {
18             return tvm.buildStateInit({
19                 contr: Contest,
20                 varInit: {_deployer: deployer},
21                 code: _codeContest
22             });
23     }
```

Chapter 12

Contract Group

Contents

12.1 Overview	70
12.2 Contract Inheritance	70
12.3 Static Variable Definitions	70
12.4 Variable Definitions	71
12.5 Constructor Definitions	71
12.5.1 Constructor	71
12.6 Public Method Definitions	71
12.6.1 Function addMember	71
12.6.2 Function getMembers	71
12.6.3 Function removeMember	72

12.1 Overview

In file `Group.sol`

12.2 Contract Inheritance

Base	
IGroup	

12.3 Static Variable Definitions

```
11  string static _name;
```

12.4 Variable Definitions

```
12 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.

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

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.

```
25 function addMember(uint128 idProposal, address member) public
26     onlyContract {
27     idProposal;
28     _members.push(member);
29 }
```

12.6.2 Function getMembers

- OK

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


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.

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

Chapter 13

Contract GroupResolver

Contents

13.1 Overview	73
13.2 Variable Definitions	73
13.3 Public Method Definitions	73
13.3.1 Function resolveGroup	73
13.4 Internal Method Definitions	74
13.4.1 Function _buildGroupState	74

13.1 Overview

In file `GroupResolver.sol`

13.2 Variable Definitions

TvmCell	_codeGroup	
		used in @16.GroupRe- solver._buildGroupState

8 TvmCell _codeGroup;

13.3 Public Method Definitions

13.3.1 Function resolveGroup

- OK

```
10     function resolveGroup(string name) public view returns (address
11         group) {
12         TvmCell state = _buildGroupState(name);
13         uint256 hashState = tvm.hash(state);
14         group = address.makeAddrStd(0, hashState);
15     }
```

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.

```
16     function _buildGroupState(string name) internal virtual view
17         returns (TvmCell) {
18         return tvm.buildStateInit({
19             contr: Group,
20             varInit: {_name: name},
21             code: _codeGroup
22         });
23     }
```

Chapter 14

Contract JuryGroup

Contents

14.1 Overview	75
14.2 Contract Inheritance	75
14.3 Static Variable Definitions	75
14.4 Variable Definitions	76
14.5 Modifier Definitions	76
14.5.1 Modifier onlyDeployer	76
14.6 Constructor Definitions	76
14.6.1 Constructor	76
14.7 Public Method Definitions	76
14.7.1 Function getMembers	76
14.7.2 Function registerMember	77
14.7.3 Function withdraw	77
14.8 Internal Method Definitions	77
14.8.1 Function _addMember	77

14.1 Overview

In file JuryGroup.sol

14.2 Contract Inheritance

IJuryGroup	
------------	--

14.3 Static Variable Definitions

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

```
12     address static _deployer;
```

14.4 Variable Definitions

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

```
15     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.

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

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.

```
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     }
```

14.7 Public Method Definitions

14.7.1 Function getMembers

- OK

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

14.7.2 Function registerMember

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

```

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

```

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`.

```

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     }

```

14.8 Internal Method Definitions

14.8.1 Function _addMember

- OK

```

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

```

Chapter 15

Contract

JuryGroupResolver

Contents

15.1 Overview	78
15.2 Variable Definitions	78
15.3 Public Method Definitions	78
15.3.1 Function resolveJuryGroup	78
15.4 Internal Method Definitions	79
15.4.1 Function _buildJuryGroupState	79

15.1 Overview

In file JuryGroupResolver.sol

15.2 Variable Definitions

```
6     TvmCell _codeJuryGroup;
```

15.3 Public Method Definitions

15.3.1 Function resolveJuryGroup

- OK

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

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

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.

```
14     function _buildJuryGroupState(string tag, address deployer)
15         internal view returns (TvmCell) {
16         return tvm.buildStateInit({
17             contr: JuryGroup,
18             varInit: {_tag: tag, _deployer: deployer},
19             code: _codeJuryGroup
20         });
21     }
```


Chapter 16

Contract Padawan

Contents

16.1 Overview	80
16.2 Contract Inheritance	81
16.3 Static Variable Definitions	81
16.4 Variable Definitions	81
16.5 Modifier Definitions	81
16.5.1 Modifier onlyOwner	81
16.6 Constructor Definitions	82
16.6.1 Constructor	82
16.7 Public Method Definitions	82
16.7.1 Function confirmVote	82
16.7.2 Function createTokenAccount	82
16.7.3 Function depositTokens	83
16.7.4 Function depositTons	83
16.7.5 Function onEstimateVotes	83
16.7.6 Function onTokenWalletDeploy	84
16.7.7 Function onTokenWalletGetBalance	85
16.7.8 Function reclaimDeposit	85
16.7.9 Function rejectVote	86
16.7.10 Function updateStatus	86
16.7.11 Function vote	87
16.8 Internal Method Definitions	88
16.8.1 Function _doReclaim	88

16.1 Overview

In file `Padawan.sol`

16.2 Contract Inheritance

Base	
IEstimateVotesCallback	

16.3 Static Variable Definitions

- OK

```
32 address static _deployer;
```

```
33 address static _owner;
```

16.4 Variable Definitions

- OK

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

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

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

```
38 uint32 _activeProposalsLength;
```

```
40 Reclaim public _reclaim;
```

16.5 Modifier Definitions

16.5.1 Modifier onlyOwner

- OK

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

16.6 Constructor Definitions

16.6.1 Constructor

- OK

```

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

```

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.

```

89     function confirmVote(
90         uint128 votes,
91         uint128 votePrice,
92         address voteProvider)
93     external onlyContract { votes;
94         optional(ActiveProposal) optActiveProposal =
95             _activeProposals.fetch(msg.sender);
96         require(optActiveProposal.hasValue(), 111);
97         uint128 activeProposalVotes = optActiveProposal.get().votes
98             ;
99
100         address balanceProvider = voteProvider == address(0) ?
101             voteProvider : _tokenAccounts[voteProvider];
102
103         if(_balances[balanceProvider].locked < (activeProposalVotes
104             * votePrice) {
105             _balances[balanceProvider].locked = (
106                 activeProposalVotes) * votePrice;
107         }
108         _owner.transfer(0, false, 64);
109     }

```

16.7.2 Function createTokenAccount

- OK

```

228     function createTokenAccount(address tokenRoot) external
229         onlyOwner {
230         require(msg.value >= DEFAULT_FEE, Errors.MSG_VALUE_TOO_LOW)
231         ;
232         require(!_tokenAccounts.exists(tokenRoot));
233
234         ITokenRoot(tokenRoot).deployEmptyWallet

```

```

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

```

16.7.3 Function depositTokens

- OK

```

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

```

16.7.4 Function depositTons

- OK

```

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

```

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.

```

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 =
68             _activeProposals.fetch(msg.sender);
69         ActiveProposal activeProposal = optActiveProposal.hasValue
70             () ? optActiveProposal.get() : ActiveProposal(
71             voteProvider, votePrice, 0);
72         if(!optActiveProposal.hasValue()) {
73             _activeProposals[msg.sender] = activeProposal;
74         }
75         optional(Balance) optBalance;
76         if(voteProvider == address(0)) {
77             optBalance = _balances.fetch(voteProvider);
78         } else {
79             optional(address) optAccount = _tokenAccounts.fetch(
80             voteProvider);
81             require(optAccount.hasValue(), 115);
82             optBalance = _balances.fetch(optAccount.get());
83         }
84         require(optBalance.hasValue(), 113);
85         require(optBalance.get().total >= (activeProposal.votes *
86             votePrice) + cost, 114);
87         _activeProposals[msg.sender].votes += votes;
88         _activeProposalsLength += 1;
89         IProposal(msg.sender).vote
90             {value: 0, flag: 64, bounce: true}
91             (_owner, choice, votes);
92     }

```

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.

```

237     function onTokenWalletDeploy(address account) public {
238         require(!_tokenAccounts.exists(msg.sender), Errors.
239             INVALID_CALLER);
240         _tokenAccounts[msg.sender] = account;
241         _balances[account] = Balance(0, 0);
242         _owner.transfer(0, false, 64);
243     }

```

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 =

```

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 }

```

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.

```

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 =
140         _activeProposals.min();
141     while (optActiveProposal.hasValue()) {
142         (address addrActiveProposal,) = optActiveProposal.get()
143         ;
144         IProposal(addrActiveProposal).queryStatus
145             {value: QUERY_STATUS_FEE, bounce: true, flag: 1}
146             ();
147         optActiveProposal = _activeProposals.next(
148             addrActiveProposal);
149     }
150 }

```

16.7.9 Function rejectVote

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

```

106 function rejectVote(uint128 votes, uint16 errorCode) external
107     onlyContract { votes; errorCode;
108     optional(ActiveProposal) optActiveProposal =
109         _activeProposals.fetch(msg.sender);
110     require(optActiveProposal.hasValue(), 112);
111     ActiveProposal activeProposal = optActiveProposal.get();
112     activeProposal.votes -= votes;
113     if (activeProposal.votes == 0) {
114         delete _activeProposals[msg.sender];
115         _activeProposalsLength -= 1;
116     }
117     _owner.transfer(0, false, 64);
118 }

```

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.

```

149     function updateStatus(ProposalState state) external
150         onlyContract {
151             optional(ActiveProposal) optActiveProposal =
152                 _activeProposals.fetch(msg.sender);
153             require(optActiveProposal.hasValue());
154             ActiveProposal activeProposal = optActiveProposal.get();
155
156             if (state >= ProposalState.Ended) {
157                 address balanceProvider = address(0);
158                 if(activeProposal.voteProvider != address(0)) {
159                     optional(address) optAccount = _tokenAccounts.fetch
160                         (activeProposal.voteProvider);
161                     require(optAccount.hasValue(), 117);
162                     balanceProvider = optAccount.get();
163                 }
164                 Balance balance = _balances[balanceProvider];
165                 if(balance.locked <= activeProposal.votes *
166                     activeProposal.votePrice) {
167                     delete _activeProposals[msg.sender];
168                     uint128 max;
169                     optional(address, ActiveProposal)
170                         optActiveProposal2 = _activeProposals.min();
171                     while (optActiveProposal2.hasValue()) {
172                         (address addrActiveProposal, ActiveProposal
173                             activeProposal2) = optActiveProposal2.get()
174                             ;
175                         if(activeProposal2.votes * activeProposal2.
176                             votePrice > max && activeProposal2.
177                             voteProvider == activeProposal.voteProvider
178                             ) {
179                             max = activeProposal2.votes *
180                                 activeProposal2.votePrice;
181                         }
182                         optActiveProposal2 = _activeProposals.next(
183                             addrActiveProposal);
184                     }
185                     _balances[balanceProvider].locked = max;
186                 } else {
187                     delete _activeProposals[msg.sender];
188                 }
189                 _activeProposalsLength -= 1;
190                 if(_reclaim.amount != 0) {
191                     balance = _balances[_reclaim.balanceProvider];
192                     if (_reclaim.amount <= balance.total - balance.
193                         locked) {
194                         _doReclaim();
195                     }
196                 }
197             }
198         }
199     }

```

16.7.11 Function vote

- OK


```
53     function vote(address proposal, bool choice, uint128 votes)
54         external onlyOwner {
55             require(msg.value >= VOTE_FEE, Errors.MSG_VALUE_TOO_LOW);
56             IProposal(proposal).estimateVotes
57                 {value: 0, flag: 64, bounce: true}
58                 (votes, choice);
59         }
```

16.8 Internal Method Definitions

16.8.1 Function _doReclaim

- OK

```
191     function _doReclaim() private inline {
192         if(_reclaim.balanceProvider == address(0)) {
193             _reclaim.returnTo.transfer(_reclaim.amount, true, 1);
194         } else {
195             ITokenWallet(_reclaim.balanceProvider).transfer
196                 {value: 0.2 ton} // refactor
197                 (_reclaim.returnTo, _reclaim.amount, 0.1 ton);
198         }
199         _balances[_reclaim.balanceProvider].total -= _reclaim.
200             amount;
201         delete _reclaim;
202         _owner.transfer(0, false, 64);
203     }
```

Chapter 17

Contract PadawanResolver

Contents

17.1 Overview	89
17.2 Variable Definitions	89
17.3 Public Method Definitions	89
17.3.1 Function resolvePadawan	89
17.4 Internal Method Definitions	90
17.4.1 Function _buildPadawanState	90

17.1 Overview

In file PadawanResolver.sol

17.2 Variable Definitions

```
8     TvmCell _codePadawan;
```

17.3 Public Method Definitions

17.3.1 Function resolvePadawan

- OK

```
10     function resolvePadawan(address owner) public view returns (
11         address addrPadawan) {
12         TvmCell state = _buildPadawanState(owner);
13         uint256 hashState = tvm.hash(state);
14         addrPadawan = address.makeAddrStd(0, hashState);
15     }
```

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.

```
16     function _buildPadawanState(address owner) internal virtual
17         view returns (TvmCell) {
18             return tvm.buildStateInit({
19                 contr: Padawan,
20                 varInit: {_deployer: address(this), _owner: owner},
21                 code: _codePadawan
22             });
23     }
```

Chapter 18

Contract Proposal

Contents

18.1 Overview	92
18.2 Contract Inheritance	92
18.3 Static Variable Definitions	92
18.4 Variable Definitions	92
18.5 Constructor Definitions	93
18.5.1 Constructor	93
18.6 Public Method Definitions	94
18.6.1 Function estimateVotes	94
18.6.2 Function getAll	94
18.6.3 Function getCurrentVotes	94
18.6.4 Function getInfo	94
18.6.5 Function getVotingResults	94
18.6.6 Function onGetMembers	95
18.6.7 Function queryStatus	95
18.6.8 Function vote	95
18.6.9 Function wrapUp	96
18.7 Internal Method Definitions	96
18.7.1 Function _buildPadawanState	96
18.7.2 Function _calculateVotes	96
18.7.3 Function _changeState	97
18.7.4 Function _finalize	97
18.7.5 Function _findInWhiteList	97
18.7.6 Function _getGroupMembers	98
18.7.7 Function _softMajority	98
18.7.8 Function _tryEarlyComplete	98
18.7.9 Function _wrapUp	99

18.1 Overview

In file `Proposal.sol`

18.2 Contract Inheritance

Base	
PadawanResolver	
GroupResolver	
IProposal	
IGroupCallback	

18.3 Static Variable Definitions

- OK

```
15     address static _deployer;
```

```
16     uint32 static _id;
```

18.4 Variable Definitions

- OK

```
18     address _client;
```

```
20     uint128 _votePrice;
```

```
21     uint128 _voteTotal;
```

```
22     address _voteProvider;
```

```
24     address[] _whiteList;
```

```
25     bool _openProposal = false;
```

```
27     ProposalInfo _proposalInfo;
```

```
29     ProposalResults _results;
```

```
30     VoteCountModel _voteCountModel;
```

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.

```

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     }

```

18.6 Public Method Definitions

18.6.1 Function estimateVotes

- OK

```

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

```

18.6.2 Function getAll

- OK

```

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

```

18.6.3 Function getCurrentVotes

- OK

```

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

```

18.6.4 Function getInfo

- OK

```

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

```

18.6.5 Function getVotingResults

- OK

```

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

```

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.

```

220     function onGetMembers(string name, address[] members) public
221         override onlyContract { name;
222             _whiteList = members;
223         }

```

18.6.7 Function queryStatus

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

```

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

```

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.

```

84     function vote(address padawanOwner, bool choice, uint128 votes)
85         external override {
86         address addrPadawan = resolvePadawan(padawanOwner);
87         uint16 errorCode = 0;
88         require(_openProposal || _findInWhiteList(padawanOwner),
89             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,
100             bounce: true}(votes, errorCode);
101     } else {
102         IPadawan(msg.sender).confirmVote{value: 0, flag: 64,
103             bounce: true}(votes, _votePrice, _voteProvider);
104         if (choice) {
105             _proposalInfo.votesFor += votes;
106         } else {
107             _proposalInfo.votesAgainst += votes;
108         }
109     }
110     _wrapUp();

```

18.6.9 Function wrapUp

- OK

```

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

```

18.7 Internal Method Definitions

18.7.1 Function _buildPadawanState

- OK

```

183     function _buildPadawanState(address owner) internal view
184         override returns (TvmCell) {
185         return tvm.buildStateInit({
186             contr: Padawan,
187             varInit: {_deployer: _deployer, _owner: owner},
188             code: _codePadawan
189         });
190     }

```

18.7.2 Function _calculateVotes

- OK

```

161     function _calculateVotes(
162         uint128 yes,
163         uint128 no

```

```

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

```

18.7.3 Function `_changeState`

- OK

```

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

```

18.7.4 Function `_finalize`

- OK

```

112     function _finalize(bool passed) private {
113         _results = ProposalResults(
114             uint32(0),
115             passed,
116             _proposalInfo.votesFor,
117             _proposalInfo.votesAgainst,
118             _voteTotal,
119             _voteCountModel,
120             uint32(now)
121         );
122
123         ProposalState state = passed ? ProposalState.Passed :
124             ProposalState.NotPassed;
125
126         _changeState(state);
127
128         IClient(address(_client)).onProposalPassed{value: 1 ton} (
129             _proposalInfo);
130     }

```

18.7.5 Function `_findInWhiteList`

- OK

```

224     function _findInWhiteList(address padawanOwner) view private
225         returns (bool) {
226         for(uint32 index = 0; index < _whiteList.length; index++) {
227             if(_whiteList[index] == padawanOwner) {
228                 return true;
229             }
230         }
231         return false;
232     }

```

18.7.6 Function `_getGroupMembers`

- OK

```

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

```

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.

```

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
176             / 2) - (_voteTotal / 10))) / (_voteTotal / 2);
177         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.

```

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     }

```

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;

```
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,
151                                     _proposalInfo.votesAgainst);
152         } else {
153             (completed, passed) = _tryEarlyComplete(_proposalInfo.
154                                                         votesFor, _proposalInfo.votesAgainst);
155         }
156
157         if (completed) {
158             _changeState(ProposalState.Ended);
159             _finalize(passed);
160         }
161     }
```

Chapter 19

Contract ProposalFactory

Contents

19.1 Overview	100
19.2 Contract Inheritance	100
19.3 Static Variable Definitions	100
19.4 Constructor Definitions	101
19.4.1 Constructor	101
19.5 Public Method Definitions	101
19.5.1 Function deployAddMemberProposal	101
19.5.2 Function deployContestProposal	101
19.5.3 Function deployRemoveMemberProposal	102

19.1 Overview

In file ProposalFactory.sol

19.2 Contract Inheritance

Base	
------	--

19.3 Static Variable Definitions

- OK

14 `address static _deployer;`

19.4 Constructor Definitions

19.4.1 Constructor

- OK

```

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

```

19.5 Public Method Definitions

19.5.1 Function deployAddMemberProposal

- OK

```

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

```

19.5.2 Function deployContestProposal

- OK

```

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

```

19.5.3 Function deployRemoveMemberProposal

- OK

```

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

```

```
72         );  
73     }
```


Chapter 20

Contract

ProposalFactoryResolver

Contents

20.1 Overview	104
20.2 Variable Definitions	104
20.3 Public Method Definitions	104
20.3.1 Function resolveProposalFactory	104
20.4 Internal Method Definitions	105
20.4.1 Function _buildProposalFactoryState	105

20.1 Overview

In file `ProposalFactoryResolver.sol`

20.2 Variable Definitions

```
6     TvmCell _codeProposalFactory;
```

20.3 Public Method Definitions

20.3.1 Function resolveProposalFactory

- OK

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

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

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.

```
14     function _buildProposalFactoryState(address deployer) internal
15         view returns (TvmCell) {
16         return tvm.buildStateInit({
17             contr: ProposalFactory,
18             varInit: {_deployer: deployer},
19             code: _codeProposalFactory
20         });
21     }
```

Chapter 21

Contract ProposalResolver

Contents

21.1 Overview	106
21.2 Variable Definitions	106
21.3 Public Method Definitions	106
21.3.1 Function resolveProposal	106
21.4 Internal Method Definitions	107
21.4.1 Function _buildProposalState	107

21.1 Overview

In file `ProposalResolver.sol`

21.2 Variable Definitions

```
6     TvmCell _codeProposal;
```

21.3 Public Method Definitions

21.3.1 Function resolveProposal

- OK

```
8     function resolveProposal(uint32 id) public view returns (
9         address addrProposal) {
10         TvmCell state = _buildProposalState(id);
11         uint256 hashState = tvm.hash(state);
12         addrProposal = address.makeAddrStd(0, hashState);
13     }
```

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.

```
14     function _buildProposalState(uint32 id) internal view returns (
15         TvmCell) {
16         return tvm.buildStateInit({
17             contr: Proposal,
18             varInit: {_deployer: address(this), _id: id},
19             code: _codeProposal
20         });
21     }
```

Chapter 22

Contract SmvRoot

Contents

22.1 Overview	108
22.2 Contract Inheritance	109
22.3 Constant Definitions	109
22.4 Variable Definitions	109
22.5 Modifier Definitions	109
22.5.1 Modifier onlyStore	109
22.6 Constructor Definitions	110
22.6.1 Constructor	110
22.7 Public Method Definitions	110
22.7.1 Function _deployProposal	110
22.7.2 Function deployGroup	111
22.7.3 Function deployPadawan	111
22.7.4 Function deployProposal	111
22.7.5 Function getStats	112
22.7.6 Function getStored	112
22.7.7 Function updateAddr	113
22.7.8 Function updateCode	113
22.8 Internal Method Definitions	113
22.8.1 Function _beforeProposalDeploy	113
22.8.2 Function _createChecks	114
22.8.3 Function _onInit	114

22.1 Overview

In file `SmvRoot.sol`

22.2 Contract Inheritance

Base	
ISmvRoot	
ISmvRootStoreCallback	
PadawanResolver	
ProposalResolver	
GroupResolver	
ProposalFactoryResolver	
Checks	

22.3 Constant Definitions

```

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

```

22.4 Variable Definitions

```

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

```

22.5 Modifier Definitions

22.5.1 Modifier onlyStore

```

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

```



```

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

```

22.7.2 Function deployGroup

- TODO

```

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

```

22.7.3 Function deployPadawan

- TODO

```

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

```

22.7.4 Function deployProposal

- TODO


```

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

```

22.7.5 Function getStats

- TODO

```

246     function getStats() public view returns (uint32
247         deployedPadawansCounter, uint32 deployedProposalsCounter) {
248         deployedPadawansCounter = _deployedPadawansCounter;
249         deployedProposalsCounter = _deployedProposalsCounter;
250     }

```

22.7.6 Function getStored

- TODO

```

230     function getStored() public view returns (
231         TvmCell codePadawan,
232         TvmCell codeProposal,
233         TvmCell codeGroup,
234         TvmCell codeProposalFactory,
235         address addrBftgRoot,
236         address proposalFactory
237     ) {
238         codePadawan = _codePadawan;
239         codeProposal = _codeProposal;

```

```

240     codeGroup = _codeGroup;
241     codeProposalFactory = _codeProposalFactory;
242     addrBftgRoot = _addrBftgRoot;
243     proposalFactory = _addrProposalFactory;
244 }

```

22.7.7 Function updateAddr

- TODO

```

123     function updateAddr(ContractAddr kind, address addr) external
124         override onlyStore {
125         require(addr != address(0));
126         if (kind == ContractAddr.BftgRoot) {
127             _addrBftgRoot = addr;
128             _passCheck(CHECK_BFTG_ROOT_ADDRESS);
129         }
130         _onInit();
131     }

```

22.7.8 Function updateCode

- TODO

```

103     function updateCode(
104         ContractCode kind,
105         TvmCell code
106     ) external override onlyStore {
107         if (kind == ContractCode.Proposal) {
108             _codeProposal = code;
109             _passCheck(CHECK_PROPOSAL);
110         } else if (kind == ContractCode.Padawan) {
111             _codePadawan = code;
112             _passCheck(CHECK_PADAWAN);
113         } else if (kind == ContractCode.Group) {
114             _codeGroup = code;
115             _passCheck(CHECK_GROUP);
116         } else if (kind == ContractCode.ProposalFactory) {
117             _codeProposalFactory = code;
118             _passCheck(CHECK_PROPOSAL_FACTORY);
119         }
120         _onInit();
121     }

```

22.8 Internal Method Definitions

22.8.1 Function _beforeProposalDeploy

- TODO

```

172     function _beforeProposalDeploy(
173         address client,
174         string title,
175         uint128 votePrice,
176         uint128 voteTotal,
177         address voteProvider,
178         address group,
179         address[] whiteList,
180         string proposalType,
181         TvmCell specific
182     ) private view {
183         TvmCell state = _buildProposalState(
184             _deployedProposalsCounter);
185         uint256 hashState = tvml.hash(state);
186         address proposal = address.makeAddrStd(0, hashState);
187         // IClient(_addrDensRoot).onProposalDeploy
188         //     {value: 1 ton, bounce: true}
189         //     (proposal, proposalType, specific);
190         this._deployProposal
191             {value: 4 ton}
192             (client, title, votePrice, voteTotal, voteProvider,
193              group, whiteList, proposalType, specific);
194     }

```

22.8.2 Function _createChecks

- TODO

```

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

```

22.8.3 Function _onInit

- TODO

```

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

```

Chapter 23

Contract SmvRootStore

Contents

23.1 Overview	115
23.2 General Minor-level Remarks	115
23.3 Contract Inheritance	116
23.4 Variable Definitions	116
23.5 Public Method Definitions	116
23.5.1 Function queryAddr	116
23.5.2 Function queryCode	116
23.5.3 Function setBftgRootAddr	117
23.5.4 Function setGroupCode	117
23.5.5 Function setPadawanCode	117
23.5.6 Function setProposalCode	117
23.5.7 Function setProposalFactoryCode	117

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-Initialization 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

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

```
11 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.

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

23.5.2 Function queryCode

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

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

23.5.3 Function setBftgRootAddr

- OK

```

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

```

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`

```

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

```

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`

```

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

```

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`

```

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

```

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`

```

22     function setProposalFactoryCode(TvmCell code) public override
23         signed {
24         _codes[uint8(ContractCode.ProposalFactory)] = code;
25     }

```