Audit of the BFTG project

By OCamlPro

August 20, 2021

Table of Major and Critical Issues

Critical issue: Administrative Take-over in BftgRoot.constructor 49 Major issue: No initialization check performed in BftgRoot.constructor 50 Critical issue: tvm.accept without check in BftgRoot.deployContest 51 Major issue: Non-reentrant in BftgRoot.registerMemberJuryGroup 52
Critical issue: No stage check in Contest.calcRewards 62
Major issue: Wrong computation in Contest.calcRewards 62
Critical issue: Missing permission checks in Contest.changeStage 62
Critical issue: Multiple revelations in Contest.reveal 64
Major issue: Unbounded storage in Contest.submit 65
Critical issue: No permission check in Contest.updateCode 65
Major issue: No gas check in Contest.updateCode 65
Critical issue: No permission check in Group.constructor 71
Critical issue: No permission check in Group.addMember 71
Critical issue: No permission check on removeMember 72
Major issue: Wrong comparison in JuryGroup.withdraw 77
Major issue: Incorrect computation in Padawan.onEstimateVotes 83
Critical issue: Can empty voting rights in Padawan.onTokenWalletDeploy 84
Critical issue: Unbounded voting rights in Padawan.onTokenWalletGetBalance 88
Critical issue: Race condition in Padawan.reclaimDeposit 85
Critical issue: No permission check on Proposal.onGetMembers 95
Critical issue: Division by 0 in Proposal. softMajority 98
Major issue: Overflow in ProposaltryEarlyComplete 98
Critical issue: Constructor for SmyRoot (fake)

Contents

1	Intr	oducti	on	11
		1.0.1	Location	11
		1.0.2	End Date	11
2	Ove	erview		12
3	Lib	rary M	odules	13
	3.1		e "BFTG.sol"	14
		3.1.1	Imports	14
	3.2	Modul	e "Errors.sol"	15
		3.2.1	Pragmas	15
		3.2.2	Contract Definitions	15
	3.3	Modul	e "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	Modul	e "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	Inte	erface]	Modules	20
_	4.1	Modul	e "IBftgRoot.sol"	21
		4.1.1	Pragmas	21
		4.1.2	Type Definitions	21
		•	4.1.2.1 Struct JuryGroupPending	21
		413	Contract Definitions	21

	4.2	Modul	le "IBftgRootStore.sol"
		4.2.1	Pragmas
		4.2.2	Type Definitions
			4.2.2.1 Enum ContractCode
			4.2.2.2 Enum ContractAddr
		4.2.3	Contract Definitions
	4.3	Modul	le "IClient.sol"
		4.3.1	Pragmas
		4.3.2	Imports
		4.3.3	Contract Definitions
	4.4		le "IGroup.sol"
		4.4.1	Pragmas
		4.4.2	Contract Definitions
	4.5		le "IJuryGroup.sol"
	1.0	4.5.1	Pragmas
		4.5.2	Type Definitions
		4.0.2	4.5.2.1 Struct Member
		4.5.3	Contract Definitions
	4.6		le "IPadawan.sol"
	4.0	4.6.1	Pragmas
		4.6.2	Imports
		4.6.2	Type Definitions
		4.0.5	4.6.3.1 Struct TipAccount
		4.6.4	Contract Definitions
	4.7		le "IProposal.sol"
	4.7	4.7.1	
		4.7.1 $4.7.2$	
		4.7.2 $4.7.3$	1
		4.7.3	v I
			1
		4 7 4	
	4.0	4.7.4	Contract Definitions
	4.8		le "ITokenRoot.sol"
		4.8.1	Pragmas
	4.0	4.8.2	Contract Definitions
	4.9		le "ITokenWallet.sol"
		4.9.1	Pragmas
		4.9.2	Contract Definitions
5	Cor	troat 1	Modules 31
J	5.1		le "Base.sol"
	5.1	5.1.1	Pragmas
		5.1.1 $5.1.2$	Imports
		5.1.2 $5.1.3$	Contract Definitions
	5.0	-	le "BftgRoot.sol"
	5.2		§
		5.2.1	Pragmas
		5.2.2	Imports

	5.2.3 Cont	ract Definitions		 		 				33
5.3	Module "Ch	ecks.sol"		 		 				34
		mas								34
		ract Definitions								34
5.4		ntest.sol"								35
		mas								35
	_	rts								35
		ract Definitions								35
5.5		ntestResolver.sol								36
		mas								36
	9	rts								36
	1	ract Definitions								36
5.6		oup.sol"								37
0.0		mas								37
	_	rts								37
	1	ract Definitions								37
5.7		oupResolver.sol"								38
0.,		mas								38
		rts								38
		ract Definitions								38
5.8		yGroup.sol"								39
0.0		mas								39
		rts								39
	-	ract Definitions								39
5.9		yGroupResolver								40
0.9	5.9.1 Pragi	mas	.501 .	 	• •	 	•	•	•	40
		rts								40
		ract Definitions								40
5.10		dawan.sol"								41
5.10		mas								41
		rts								41
		Definitions								41
	5.10.5 Type 5.10.5									41
	5.10.3 5.10.3									41
	5.10.3									41
	5.10.3 5.10.3									41
		ract Definitions								42
E 11		ract Demittions dawanResolver.se								43
3.11				 		 	•			
	9	mas								43
	•	rts								43
F 10		ract Definitions								43
5.12		oposal.sol"								44
	_	mas								44
	_	rts					•			44
	5.12.3 Confi	ract Definitions		 		 				44

6	Cor	ntract Base 45
	6.1	Overview
	6.2	Constant Definitions
	6.3	Modifier Definitions
		6.3.1 Modifier signed
		6.3.2 Modifier accept
		6.3.3 Modifier onlyContract
		6.3.4 Modifier onlyMe
7	Cor	ntract BftgRoot 48
•	7.1	Overview
	7.2	Contract Inheritance
	7.3	Constant Definitions
	7.4	Variable Definitions
	7.5	Modifier Definitions
	1.0	7.5.1 Modifier onlyStore
	7.6	Constructor Definitions
	1.0	7.6.1 Constructor
	7.7	Public Method Definitions
		7.7.1 OnBounce function
		7.7.2 Function deployContest
		7.7.3 Function deployJuryGroup
		7.7.4 Function getMembersCallback
		7.7.5 Function getStored
		7.7.6 Function registerMemberJuryGroup
		7.7.7 Function updateAddr
		7.7.8 Function updateCode
	7.8	Internal Method Definitions
	•••	7.8.1 Function _createChecks
		7.8.2 Function _onInit
	~	
8		ntract BftgRootStore 54
	8.1	Overview
	8.2	General Minor-level Remarks
	8.3	Contract Inheritance
	8.4	Variable Definitions
	8.5	Public Method Definitions
		8.5.1 Function queryAddr
		8.5.2 Function queryCode
		8.5.3 Function setContestCode
		8.5.4 Function set. Jury Group Code 56

9	Con	tract Checks 57
	9.1	Overview
	9.2	Variable Definitions
	9.3	Modifier Definitions
		9.3.1 Modifier checksEmpty
	9.4	Internal Method Definitions
		9.4.1 Function _isCheckListEmpty
		9.4.2 Function _passCheck
10	Con	tract Contest 59
	10.1	Overview
	10.2	Contract Inheritance
		Constant Definitions
		Static Variable Definitions
		Variable Definitions
	10.6	Constructor Definitions
		10.6.1 Constructor
	10.7	Public Method Definitions 61
		10.7.1 OnBounce function
		10.7.2 Function calcRewards
		10.7.3 Function changeStage
		10.7.4 Function claimPartisipantReward 62
		10.7.5 Function getHiddenVotesByAddress 63
		10.7.6 Function getMembersCallback 63
		10.7.7 Function hashVote
		10.7.8 Function reveal
		10.7.9 Function stakePartisipantReward
		10.7.10 Function submit
		10.7.11 Function updateAddr
		10.7.12 Function updateCode
		10.7.13 Function vote
	10.8	Internal Method Definitions
	10.0	10.8.1 Function _calcPointValue
		10.8.2 Function _changeStage
		10.8.3 Function _createChecks
		10.8.4 Function onlinit
11		tract ContestResolver 68
		Overview
		Variable Definitions
	11.3	Public Method Definitions
		11.3.1 Function resolveContest
	11.4	Internal Method Definitions
		11.4.1 Function buildContestState 69

12	Con	tract 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		±	7 3
		Overview	73
		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
11	C	tt IC	75
14		tract JuryGroup Overview	75
		Contract Inheritance	75
		Static Variable Definitions	75
		Variable Definitions	76
		Modifier Definitions	76
	14.5	14.5.1 Modifier onlyDeployer	76
	116	Constructor Definitions	76
	14.0	14.6.1 Constructor	76
	147	Public Method Definitions	76
	14.7		76
		14.7.1 Function getMembers	77
		14.7.2 Function registerMember	
	140		77
	14.8	Internal Method Definitions	77
		14.8.1 Function _addMember	77
15	Con	tract JuryGroupResolver	7 8
	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	Con		30
	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
		1	83
		16.7.4 Function depositTons	83
			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
1 =	a	D 1 D 1	20
17			89
17	17.1	Overview	89
17	$17.1 \\ 17.2$	Overview	89 89
17	$17.1 \\ 17.2$	Overview 8 Variable Definitions 8 Public Method Definitions 8	89 89 89
17	17.1 17.2 17.3	Overview8Variable Definitions8Public Method Definitions817.3.1 Function resolvePadawan8	89 89 89
17	17.1 17.2 17.3	Overview8Variable Definitions8Public Method Definitions817.3.1 Function resolvePadawan8Internal Method Definitions9	89 89 89 89
17	17.1 17.2 17.3	Overview8Variable Definitions8Public Method Definitions817.3.1 Function resolvePadawan8Internal Method Definitions9	89 89 89
	17.1 17.2 17.3 17.4	Overview8Variable Definitions8Public Method Definitions817.3.1 Function resolvePadawan8Internal Method Definitions917.4.1 Function buildPadawanState9	89 89 89 89 90
	17.1 17.2 17.3 17.4 Con	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9	89 89 89 89 90 90
	17.1 17.2 17.3 17.4 Con 18.1	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9	89 89 89 89 90 90
	17.1 17.2 17.3 17.4 Con 18.1 18.2	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9	89 89 89 89 90 90 91 92
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9	89 89 89 89 90 90 91 92 92
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9	89 89 89 89 90 90 91 92 92 92
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9 Constructor Definitions 9	89 89 89 89 90 90 91 92 92 92 93
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4 18.5	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9 Constructor Definitions 9 18.5.1 Constructor 9	89 89 89 89 90 90 91 92 92 92 93
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4 18.5	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9 Constructor Definitions 9 18.5.1 Constructor 9 Public Method Definitions 9	89 89 89 89 90 90 91 92 92 92 93 93
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4 18.5	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9 Constructor Definitions 9 18.5.1 Constructor 9 Public Method Definitions 9 18.6.1 Function estimateVotes 9	89 89 89 89 90 90 91 92 92 92 93 94
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4 18.5	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9 Constructor Definitions 9 18.5.1 Constructor 9 Public Method Definitions 9 18.6.1 Function estimateVotes 9 18.6.2 Function getAll 9	89 89 89 89 90 91 92 92 92 93 94 94
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4 18.5	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9 Constructor Definitions 9 18.5.1 Constructor 9 Public Method Definitions 9 18.6.1 Function estimateVotes 9 18.6.2 Function getAll 9 18.6.3 Function getCurrentVotes 9	89 89 89 89 90 90 91 92 92 92 93 94 94 94
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4 18.5	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9 Constructor Definitions 9 18.5.1 Constructor 9 Public Method Definitions 9 18.6.1 Function estimateVotes 9 18.6.2 Function getAll 9 18.6.3 Function getCurrentVotes 9 18.6.4 Function getInfo 9	89 89 89 89 90 91 92 92 92 93 94 94 94
	17.1 17.2 17.3 17.4 Con 18.1 18.2 18.3 18.4 18.5	Overview 8 Variable Definitions 8 Public Method Definitions 8 17.3.1 Function resolvePadawan 8 Internal Method Definitions 9 17.4.1 Function buildPadawanState 9 tract Proposal 9 Overview 9 Contract Inheritance 9 Static Variable Definitions 9 Variable Definitions 9 Constructor Definitions 9 18.5.1 Constructor 9 Public Method Definitions 9 18.6.1 Function estimateVotes 9 18.6.2 Function getAll 9 18.6.3 Function getUrrentVotes 9 18.6.4 Function getInfo 9 18.6.5 Function getVotingResults 9	89 89 89 89 90 90 91 92 92 92 93 94 94 94

		18.6.7 Function queryStatus	5
		18.6.8 Function vote	5
		18.6.9 Function wrapUp	3
	18.7	Internal Method Definitions	ŝ
		18.7.1 Function _buildPadawanState 90	6
		18.7.2 Function _calculateVotes	6
		18.7.3 Function _changeState 9'	7
		18.7.4 Function _finalize	7
		18.7.5 Function _findInWhiteList 9'	7
		18.7.6 Function _getGroupMembers 98	3
		18.7.7 Function softMajority	3
		18.7.8 Function _tryEarlyComplete	3
		18.7.9 Function _wrapUp	9
19	Con	tract ProposalFactory 100	0
		Overview	Э
	19.2	Contract Inheritance	Э
	19.3	Static Variable Definitions	Э
	19.4	Constructor Definitions	1
		19.4.1 Constructor	1
	19.5	Public Method Definitions	1
		19.5.1 Function deployAddMemberProposal	1
		19.5.2 Function deployContestProposal	1
		19.5.3 Function deploy RemoveMemberProposal 10:	2
20	Con	tract ProposalFactoryResolver 104	1
	20.1	Overview	4
	20.2	Variable Definitions	4
	20.3	Public Method Definitions	4
		20.3.1 Function resolveProposalFactory	4
	20.4	Internal Method Definitions	õ
		20.4.1 Function _buildProposalFactoryState 108	5
21	Con	tract ProposalResolver 100	3
	21.1	Overview	3
	21.2	Variable Definitions	3
	21.3	Public Method Definitions	3
		21.3.1 Function resolveProposal	3
	21.4	Internal Method Definitions	
		21.4.1 Function _buildProposalState 10 $^{\prime}$	7
22	Con	tract SmvRoot 108	3
	22.1	Overview	3
	22.2	Contract Inheritance	9
		Constant Definitions	
		Variable Definitions	

	22.5	Modifier Definitions
		22.5.1 Modifier onlyStore
	22.6	Constructor Definitions
		22.6.1 Constructor
	22.7	Public Method Definitions
		22.7.1 Function _deployProposal
		22.7.2 Function deployGroup
		22.7.3 Function deployPadawan
		22.7.4 Function deploy Proposal
		22.7.5 Function getStats
		22.7.6 Function getStored
		22.7.7 Function updateAddr
		22.7.8 Function updateCode
	22.8	Internal Method Definitions
		22.8.1 Function _beforeProposalDeploy
		22.8.2 Function $_$ createChecks
		22.8.3 Function $_onInit$
23		tract SmvRootStore 115
		Overview
		General Minor-level Remarks
		Contract Inheritance
		Variable Definitions
	23.5	Public Method Definitions
		23.5.1 Function queryAddr
		23.5.2 Function queryCode
		23.5.3 Function setBftgRootAddr
		23.5.4 Function setGroupCode
		23.5.5 Function set PadawanCode
		23.5.6 Function setProposalCode
		23.5.7 Function setProposalFactoryCode 117

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

Overview

Library Modules

Module "BFTG.sol" 3.1

3.1.1 Imports

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

3.2 Module "Errors.sol"

3.2.1 Pragmas

ton \mid -solidity $>=0.37.0$	ton
---------------------------------	-----

3.2.2 Contract Definitions

• Errors

3.3 Module "Glossary.sol"

3.3.1 Pragmas

```
ton \mid -solidity >= 0.36.0
```

3.3.2 Type Definitions

3.3.2.1 Enum VoteCountModel

Undefined	
Majority	
SoftMajority	
SuperMajority	
Other	
Reserved	
Last	

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

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

3.4 Module "IContest.sol"

3.4.1 Pragmas

```
ton \mid -solidity >= 0.42.0
```

3.4.2 Type Definitions

3.4.2.1 Enum ContestStage

Undefined	
New	
Underway	
Voting	
Reveal	
Rank	
Reward	
Finish	
Last	

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

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 }
```

Interface Modules

4.1 Module "IBftgRoot.sol"

4.1.1 Pragmas

```
ton \mid -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

 $\bullet \ \ IBftgRoot$

${\bf 4.2 \quad Module \ "IBftgRootStore.sol"}$

4.2.1 Pragmas

JuryGroup

```
ton \mid -solidity >= 0.42.0
```

4.2.2 Type Definitions

4.2.2.1 Enum ContractCode

```
Contest

3 enum ContractCode {
    JuryGroup,
    Contest
```

4.2.2.2 Enum ContractAddr

```
empty

8 enum ContractAddr {
9 empty
10 }
```

4.2.3 Contract Definitions

- IBftgRootStore
- $\bullet \ \ IBftgRootStoreCallback$

4.3 Module "IClient.sol"

4.3.1 Pragmas

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

4.3.2 Imports

./IProposal.sol	
/Glossary.sol	

4.3.3 Contract Definitions

• IClient

4.4 Module "IGroup.sol"

4.4.1 Pragmas

4.4.2 Contract Definitions

- IGroup
- $\bullet \ \ IGroup Callback$

4.5 Module "IJuryGroup.sol"

4.5.1 Pragmas

```
ton \mid -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

- $\bullet \ \ IJuryGroup$
- $\bullet \ \ IJuryGroupCallback$

4.6 Module "IPadawan.sol"

4.6.1 Pragmas

```
ton \mid -solidity >= 0.36.0
```

4.6.2 Imports

```
./IProposal.sol
```

4.6.3 Type Definitions

4.6.3.1 Struct TipAccount

```
addr address
balance uint128
```

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

4.6.4 Contract Definitions

• IPadawan

4.7 Module "IProposal.sol"

4.7.1 Pragmas

```
ton \mid -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
- \bullet IEstimateVotesCallback

4.8 Module "ITokenRoot.sol"

4.8.1 Pragmas

ton -solidity $>= 0.42.0$

4.8.2 Contract Definitions

• ITokenRoot

4.9 Module "ITokenWallet.sol"

4.9.1 Pragmas

ton \mid -solidity $>= 0.42.0$

4.9.2 Contract Definitions

• ITokenWallet

Contract Modules

5.1 Module "Base.sol"

5.1.1 Pragmas

ton	-solidity $>= 0.42.0$	
msgValue	2e7	

5.1.2 Imports

./Errors.sol

5.1.3 Contract Definitions

• Base

5.2 Module "BftgRoot.sol"

5.2.1 Pragmas

ton	-solidity $>=0.36.0$	
AbiHeader	expire	
AbiHeader	time	

5.2.2 Imports

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

5.2.3 Contract Definitions

• BftgRoot

5.3 Module "Checks.sol"

5.3.1 Pragmas

on	-solidity $>= 0.42.0$	
/11	-5011010y > -0.42.0	

5.3.2 Contract Definitions

• Checks

5.4 Module "Contest.sol"

5.4.1 Pragmas

ton -solidity $>= 0.43.0$

5.4.2 Imports

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

5.4.3 Contract Definitions

• Contest

5.5 Module "ContestResolver.sol"

5.5.1 Pragmas

ton	-solidity $>= 0.43.0$	
AbiHeader	expire	
AbiHeader	time	

5.5.2 Imports

../Contest.sol

5.5.3 Contract Definitions

 $\bullet \ \ ContestResolver$

5.6 Module "Group.sol"

5.6.1 Pragmas

ton	-solidity $>= 0.36.0$	
AbiHeader	expire	
AbiHeader	time	

5.6.2 Imports

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

5.6.3 Contract Definitions

• Group

${\bf 5.7}\quad {\bf Module~"Group Resolver.sol"}$

5.7.1 Pragmas

ton	-solidity $>= 0.36.0$	
AbiHeader	expire	
AbiHeader	time	

5.7.2 Imports

../Group.sol

5.7.3 Contract Definitions

 $\bullet \ \ {\rm GroupResolver}$

5.8 Module "JuryGroup.sol"

5.8.1 Pragmas

ton \mid -solidity >= 0.36.0

5.8.2 Imports

./interfaces/IJuryGroup.sol

5.8.3 Contract Definitions

• JuryGroup

5.9 Module "JuryGroupResolver.sol"

5.9.1 Pragmas

ton \mid -solidity >= 0.42.0

5.9.2 Imports

../JuryGroup.sol

5.9.3 Contract Definitions

• JuryGroupResolver

5.10 Module "Padawan.sol"

5.10.1 Pragmas

ton	-solidity $>= 0.36.0$	
AbiHeader	expire	
AbiHeader	time	

5.10.2 Imports

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

5.10.3 Type Definitions

5.10.3.1 Struct PadawanData

ownerAddress	address	
addr	address	

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

5.10.3.2 Struct Balance

tota	al	uint128	
lock	ĸed	uint128	

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

5.10.3.3 Struct ActiveProposal

voteProvider	address	
votePrice	uint128	
votes	uint128	

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

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

5.10.3.4 Struct Reclaim

balanceProvider	address	
amount	uint128	
returnTo	address	

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

5.10.4 Contract Definitions

• Padawan

5.11 Module "PadawanResolver.sol"

5.11.1 Pragmas

ton	-solidity $>= 0.36.0$	
AbiHeader	expire	
AbiHeader	time	

5.11.2 Imports

../Padawan.sol

5.11.3 Contract Definitions

• PadawanResolver

5.12 Module "Proposal.sol"

5.12.1 Pragmas

ton	-solidity $>= 0.36.0$	
AbiHeader	expire	
AbiHeader	time	

5.12.2 Imports

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

5.12.3 Contract Definitions

• Proposal

Chapter 6

Contract Base

Contents		
6.1	Ove	rview
6.2	Con	stant Definitions 45
6.3	Mod	lifier Definitions
	6.3.1	Modifier signed
	6.3.2	Modifier accept
	6.3.3	Modifier onlyContract 47
	6.3.4	Modifier onlyMe

6.1 Overview

In file Base.sol

6.2 Constant Definitions

```
uint64 constant DEPLOY_PAY = DEPLOY_FEE + PROCESS_FEE;
17
       uint64 constant DEPLOY_PROPOSAL_FEE = 3 ton;
       uint64 constant DEPLOY_PROPOSAL_PAY = DEPLOY_PROPOSAL_FEE +
19
           PROCESS_FEE;
       uint64 constant DEPOSIT_TONS_FEE = 1 ton;
20
       uint64 constant DEPOSIT_TONS_PAY
                                          = DEPOSIT_TONS_FEE +
           PROCESS_FEE;
       uint64 constant DEPOSIT_TOKENS_FEE = 0.5 ton +
22
           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;
       uint64 constant DEF_RESPONSE_VALUE = 0.03 ton;
29
       uint64 constant DEF_COMPUTE_VALUE = 0.2 ton;
```

6.3 Modifier Definitions

6.3.1 Modifier signed

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

6.3.2 Modifier accept

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

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

6.3.3 Modifier onlyContract

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

6.3.4 Modifier onlyMe

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

Chapter 7

Contract BftgRoot

Contents			
7.1	Ove	rview	48
7.2	Con	tract Inheritance	49
7.3	Con	stant Definitions	49
7.4	\mathbf{Vari}	able Definitions	49
7.5	Mod	lifier Definitions	49
	7.5.1	Modifier onlyStore	49
7.6	Con	structor Definitions	49
	7.6.1	Constructor	49
7.7	Pub	lic 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	Inte	rnal 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

```
uint8 constant CHECK_CONTEST_CODE = 1;
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.constructo

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 {
             if (msg.sender == address(0)) {
37
38
                  require(msg.pubkey() == tvm.pubkey(), Errors.
                      ONLY_SIGNED);
39
40
             require(addrBftgRootStore != address(0), Errors.
                 STORE_UNDEFINED);
41
             tvm.accept();
42
             _addrBftgRootStore = addrBftgRootStore;
43
             {\tt IBftgRootStore}\,(\,{\tt addrBftgRootStore})\,.\,{\tt queryCode}
44
45
                  {value: 0.2 ton, bounce: true}
46
                  (ContractCode.Contest);
             {\tt IBftgRootStore}\,(\,{\tt addrBftgRootStore})\,.\,{\tt queryCode}
47
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 {
            if(_juryGroupPendings.exists(msg.sender)) {
84
85
                address[]
86
                deployJuryGroup(_juryGroupPendings[msg.sender].tag, _);
87
                \verb|this|.registerMemberJuryGroup|
                     {value: 0, bounce: false, flag: 64}
88
                     (_juryGroupPendings[msg.sender].tag,
89
                         _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.

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.

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.

7.7.5 Function getStored

• OK

```
function getStored() public view returns (
    TvmCell codeContest,

function getStored() public view returns (
    TvmCell codeContest,

function getStored() public view returns (
    TvmCell codeContest,

    codeJuryGroup

codeJuryGroup;

function getStored() public view returns (
    TvmCell codeContest,
    codeJuryGroup

codeJuryGroup = _codeJuryGroup;

function getStored() public view returns (
    TvmCell codeContest,
    codeJuryGroup

codeJuryGroup = _codeJuryGroup;

function getStored() public view returns (
    TvmCell codeContest,
    TvmCell codeContest,
    codeJuryGroup

codeJuryGroup = _codeJuryGroup;

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

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

function getStored() public view returns (
    codeContest = _codeContest;
    codeJuryGroup = _codeJuryGroup;
    codeContest = _codeContest = _cod
```

7.7.6 Function registerMemberJuryGroup

Major issue: Non-reentrant in BftgRoot.registerMemberJuryGroup If several registerMemberJuryGroup messages are sent together for the same JuryGroup, only the last one is taken into account, in getMembersCallback. This issue might lead to missing members, or to balance problems, given that

- multiple messages sent to JuryGroup.registerMember seems to be way to increase the balance for a particular member. Fix: either the contract should deal with multiple registration at the same time, or registerMemberJuryGroup should immediately fail if a registration is already in progress for the same group.
- Minor issue (readability): an integer is used as an error. Fix: a constant should be defined instead.

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

7.7.7 Function updateAddr

• OK

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

7.7.8 Function updateCode

OK

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

${\bf Contents}$		
8.1	Ove	rview
8.2	Gen	eral Minor-level Remarks 54
8.3	Con	tract Inheritance
8.4	Vari	able Definitions
8.5	Pub	lic Method Definitions
	8.5.1	Function queryAddr
	8.5.2	Function queryCode
	8.5.3	Function setContestCode
	8.5.4	Function setJuryGroupCode

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.BftgRoot-
		Store.queryAddr
mapping (uint8 $=>$ TvmCell)	_codes	
		assigned in @1.BftgRoot-
		Store.setJuryGroupCode
		used in @1.BftgRoot-
		Store.setJuryGroupCode
		assigned in @1.BftgRoot-
		Store.setContestCode
		used in @1.BftgRoot-
		Store.setContestCode
		used in @1.BftgRoot-
		Store.queryCode

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

8.5 Public Method Definitions

8.5.1 Function queryAddr

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

8.5.2 Function queryCode

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

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

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

Chapter 9

Contract Checks

Contents

0 011001100			
9.1	Ove	erview	57
9.2	Vari	iable Definitions	57
9.3	Mod	difier Definitions	57
	9.3.1	Modifier checksEmpty	57
9.4	Inte	ernal 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 <code>BftgRoot</code>. 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.

9.4 Internal Method Definitions

9.4.1 Function _isCheckListEmpty

• OK

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

9.4.2 Function $_{-}$ passCheck

 \bullet OK

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

Chapter 10

Contract Contest

Contents	
10.1 Overview	
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 $\dots \dots \dots$	
10.7.3 Function changeStage 62	
10.7.4 Function claimPartisipantReward 62	
$10.7.5 \;\; \text{Function getHiddenVotesByAddress} \;\; \dots \;\; \dots \;\; 63$	
10.7.6 Function getMembersCallback 63	
10.7.7 Function hashVote	
10.7.8 Function reveal	
10.7.9 Function stake PartisipantReward 64	
10.7.10 Function submit	
10.7.11 Function updateAddr	
10.7.12 Function updateCode	
10.7.13 Function vote	
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	

10.1 Overview

In file Contest.sol

10.2 Contract Inheritance

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

JuryGroupResolver	
IJuryGroupCallback	
IBftgRootStoreCallback	
Checks	

10.3 Constant Definitions

OK

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

10.4 Static Variable Definitions

• OK

```
address static _deployer;
```

10.5 Variable Definitions

OK

```
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;
```

10.6 Constructor Definitions

10.6.1 Constructor

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

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

10.7 Public Method Definitions

10.7.1 OnBounce function

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

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

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 twm.accept.

10.7.4 Function claimPartisipantReward

• Minor issue: fix spelling of participant instead of partisipant.

10.7.5 Function getHiddenVotesByAddress

OK

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

10.7.6 Function getMembersCallback

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

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

10.7.7 Function hashVote

OK

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++) {</pre>
159
                 uint oldHash = _juryHiddenVotes[msg.sender][revealVotes
                     [i].submissionId].hash;
160
                 uint newHash = hashVote(revealVotes[i].submissionId,
                     revealVotes[i].score, revealVotes[i].comment);
161
                 require(oldHash == newHash, 106);
162
                 _submissionVotes[revealVotes[i].submissionId].push(Vote
                     (msg.sender, revealVotes[i].score, revealVotes[i].
                     comment)):
163
164
            msg.sender.transfer(0, true, 64);
165
```

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

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.

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.

10.7.13 Function vote

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

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

10.8 Internal Method Definitions

10.8.1 Function _calcPointValue

• OK

10.8.2 Function _changeStage

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

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

_stage = stage;
}</pre>
```

10.8.3 Function _createChecks

• OK

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

10.8.4 Function onInit

• TODO

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

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

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

11.4 Internal Method Definitions

11.4.1 Function _buildContestState

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

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

Chapter 12

Contract Group

Contents	
12.1 Overview	70
12.2 Contract Inheritance	7 0
12.3 Static Variable Definitions	7 0
12.4 Variable Definitions	7 1
12.5 Constructor Definitions	71
12.5.1 Constructor	71
12.6 Public Method Definitions	71
12.6.1 Function addMember	71

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.

12.6 Public Method Definitions

12.6.1 Function addMember

Critical issue: No permission check in Group.addMember

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

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

12.6.2 Function getMembers

OK

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

12.6.3 Function removeMember

Critical issue: No permission check on removeMember

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

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

Contract GroupResolver

Contents

13.1 Overview	3
13.2 Variable Definitions	3
13.3 Public Method Definitions	3
13.3.1 Function resolveGroup	3
13.4 Internal Method Definitions	4
13.4.1 Function _buildGroupState	4

13.1 Overview

In file ${\tt GroupResolver.sol}$

13.2 Variable Definitions

TvmCell	₋codeGroup			
		used	in	@16.GroupRe-
		solver.	.buildGr	oupState

8 TvmCell _codeGroup;

13.3 Public Method Definitions

13.3.1 Function resolveGroup

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.

Contract JuryGroup

Contents	
14.1 Overview	7 5
14.2 Contract Inheritance	7 5
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

```
string static public _tag;

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.

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

14.6 Constructor Definitions

14.6.1 Constructor

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

```
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     }</pre>
```

14.7 Public Method Definitions

14.7.1 Function getMembers

OK

14.7.2 Function registerMember

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

14.7.3 Function withdraw

Major issue: Wrong comparison in JuryGroup.withdraw

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

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

14.8 Internal Method Definitions

14.8.1 Function _addMember

Contract JuryGroupResolver

Contents

Contonio	
15.1 Overview	
15.2 Variable Definitions	
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

```
function resolveJuryGroup(string tag, address deployer) public
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.

Contract Padawan

Contents	
16.1 Overview	
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	
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 on Estimate Votes 83	
16.7.6 Function on Token Wallet Deploy 84	
16.7.7 Function onTokenWalletGetBalance 85	
16.7.8 Function reclaimDeposit 85	
16.7.9 Function rejectVote 86	
16.7.10 Function updateStatus	
16.7.11 Function vote	
16.8 Internal Method Definitions	
16.8.1 Function _doReclaim	

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

```
mapping(address => Balance) public _balances;
mapping(address => address) public _tokenAccounts;
mapping(address => ActiveProposal) public _activeProposals;
uint32 _activeProposalsLength;
Reclaim public _reclaim;
```

16.5 Modifier Definitions

16.5.1 Modifier onlyOwner

OK

16.6 Constructor Definitions

16.6.1 Constructor

• OK

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

16.7.2 Function createTokenAccount

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

ITokenRoot(tokenRoot).deployEmptyWallet
```

16.7.3 Function deposit Tokens

OK

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

16.7.4 Function depositTons

OK

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

16.7.5 Function on Estimate Votes

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

16.7.6 Function on Token Wallet Deploy

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.

16.7.7 Function on Token Wallet Get Balance

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 =

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);
            require(msg.value >= QUERY_STATUS_FEE *
120
                 _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();
             require(amount <= balance.total, Errors.NOT_ENOUGH_VOTES);</pre>
130
131
            require(returnTo != address(0), 132);
132
133
             _reclaim = Reclaim(balanceProvider, amount, returnTo);
134
```

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

16.7.9 Function rejectVote

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

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

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

16.7.11 Function vote

OK

16.8 Internal Method Definitions

16.8.1 Function _doReclaim

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

Contract PadawanResolver

$\underline{\textbf{Contents}}$

17.1 Overview	,
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

17.4 Internal Method Definitions

17.4.1 Function _buildPadawanState

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

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

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 on Get Members	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
	97
18.7.4 Function finalize	97
	97
18.7.6 Function _getGroupMembers	98
18.7.7 Function softMajority	98
- J - J - F	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

 \bullet OK

```
15 address static _deployer;

16 uint32 static _id;
```

18.4 Variable Definitions

```
address _client;
20
   uint128 _votePrice;
   uint128 _voteTotal;
21
   address _voteProvider;
22
   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.

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

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

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
         override onlyContract { name;
221          _whiteList = members;
222    }
```

18.6.7 Function queryStatus

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

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

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

18.6.9 Function wrapUp

• OK

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

18.7 Internal Method Definitions

18.7.1 Function _buildPadawanState

• OK

```
function _buildPadawanState(address owner) internal view
override returns (TvmCell) {

return tvm.buildStateInit({

contr: Padawan,

varInit: {_deployer: _deployer, _owner: owner},

code: _codePadawan

});

188
});
```

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

18.7.4 Function _finalize

OK

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

18.7.5 Function _findInWhiteList

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

18.7.6 Function _getGroupMembers

• OK

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

18.7.7 Function softMajority

Critical issue: Division by 0 in Proposal._softMajority

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

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

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.

```
{\tt function} \  \  {\tt \_tryEarlyComplete} \  (
130
131
              uint128 yes,
132
              uint128 no
133
          ) private view returns (bool, bool) {
134
               (bool completed, bool passed) = (false, false);
              if (yes * 2 > _voteTotal) {
    completed = true;
135
136
137
                   passed = true;
              } else if(no * 2 >= _voteTotal) {
138
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
             if (now > _proposalInfo.end) {
148
149
                 completed = true;
150
            ____proposalInfo.votesAgainst);
} else {
                 passed = _calculateVotes(_proposalInfo.votesFor,
151
                 (completed, passed) = _tryEarlyComplete(_proposalInfo.
152
                     votesFor, _proposalInfo.votesAgainst);
            }
153
154
155
             if (completed) {
                 _changeState(ProposalState.Ended);
156
157
                 _finalize(passed);
158
159
```

Contract ProposalFactory

Contents
19.1 Overview
19.2 Contract Inheritance
19.3 Static Variable Definitions
19.4 Constructor Definitions
19.4.1 Constructor
19.5 Public Method Definitions
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

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

19.5.2 Function deployContestProposal

```
20
        function deployContestProposal(
21
            address client,
22
            string title,
23
            address group,
24
            {\tt ContestProposalSpecific specific}
25
       ) external view onlyContract {
            require(msg.value >= DEPLOY_PROPOSAL_PAY + 1 ton);
26
27
            TvmBuilder b;
28
            b.store(specific);
29
            TvmCell cellSpecific = b.toCell();
30
            address[] arr;
31
            {\tt 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
                );
```

19.5.3 Function deployRemoveMemberProposal

OK

```
46
       function deployRemoveMemberProposal(
47
           address client,
48
           string title,
           uint128 votePrice,
49
50
           uint128 voteTotal,
51
            address voteProvider,
52
            address group,
           address[] whiteList,
53
           RemoveMemberProposalSpecific specific
       ) external view onlyContract {
55
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
                    client,
63
64
                    title,
65
                    votePrice,
66
                    voteTotal,
67
                    voteProvider,
68
                    group,
69
                    whiteList,
70
                    'remove-member',
71
                    cellSpecific
```

72); 73 }

Contract ProposalFactoryResolver

Contents

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

20.1 Overview

 ${\rm In} \ {\rm file} \ {\tt ProposalFactoryResolver.sol}$

20.2 Variable Definitions

```
6 TvmCell _codeProposalFactory;
```

20.3 Public Method Definitions

20.3.1 Function resolveProposalFactory

```
function resolveProposalFactory(address deployer) public view
    returns (address addrProposalFactory) {
    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.

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

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

 ${\rm In} \,\, {\rm file} \,\, {\tt ProposalResolver.sol}$

21.2 Variable Definitions

```
6 TvmCell _codeProposal;
```

21.3 Public Method Definitions

21.3.1 Function resolveProposal

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.

Contract SmvRoot

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

22.1 Overview

In file SmvRoot.sol

22.2 Contract Inheritance

Base	
ISmvRoot	
ISmvRootStoreCallback	
PadawanResolver	
ProposalResolver	
GroupResolver	
ProposalFactoryResolver	
Checks	

22.3 Constant Definitions

```
uint8 constant CHECK_PROPOSAL = 1;

uint8 constant CHECK_PADAWAN = 2;

uint8 constant CHECK_GROUP = 4;

uint8 constant CHECK_PROPOSAL_FACTORY = 8;

uint8 constant CHECK_BFTG_ROOT_ADDRESS = 16;
```

22.4 Variable Definitions

```
address _addrSmvRootStore;

address _addrBftgRoot;

address _addrProposalFactory;

bool public _inited = false;

uint32 _deployedPadawansCounter;

uint32 _deployedProposalsCounter;
```

22.5 Modifier Definitions

22.5.1 Modifier onlyStore

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

22.6 Constructor Definitions

22.6.1 Constructor

Critical issue: Constructor for SmvRoot (fake)

loren ipsum loren

loren ipsum loren

• TODO

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

22.7 Public Method Definitions

22.7.1 Function _deployProposal

```
194 function _deployProposal(
195 address client,
196 string title,
```

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

22.7.2 Function deployGroup

• TODO

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

22.7.3 Function deployPadawan

• TODO

22.7.4 Function deployProposal

```
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);
             require(msg.value >= DEPLOY_PROPOSAL_FEE);
155
156
             TvmBuilder b;
             b.store(specific);
157
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

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

22.7.6 Function getStored

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

22.7.7 Function updateAddr

• TODO

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

22.7.8 Function updateCode

• TODO

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

22.8 Internal Method Definitions

22.8.1 Function _beforeProposalDeploy

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

22.8.2 Function _createChecks

• TODO

22.8.3 Function onInit

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

Contents

Contract SmvRootStore

23.1	Overview
23.2	General Minor-level Remarks
23.3	Contract Inheritance
23.4	Variable Definitions

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

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

23.3 Contract Inheritance

Base	
ISmvRootStore	

23.4 Variable Definitions

```
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
            : 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
            : 64, bounce: false}(kind, code);
34    }
```

23.5.3 Function setBftgRootAddr

• OK

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

23.5.4 Function setGroupCode

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

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

23.5.5 Function setPadawanCode

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

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

23.5.6 Function setProposalCode

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

23.5.7 Function setProposalFactoryCode

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

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