Audit of the DENS-SMV Project

By OCamlPro

August 20, 2021

Table of Major and Critical Issues

Critical issue: Administrative Take-over in `Demiurge.constructor` 12
Major issue: No initialization check performed in `Demiurge.constructor` 12
Critical issue: No permission check in `Demiurge.getTotalDistributedCb` 15
Critical issue: No value check in `Demiurge.getTotalDistributedCb` 15

Critical issue: Unlimited voting rights in `Padawan.vote` 27
Major issue: Infinite locking of deposits in `Padawan.vote` . . . 27

Critical issue: Division by 0 in `Proposal._softMajority` 36
Major issue: Overflow in `Proposal._tryEarlyComplete` 37

Contents

1 Introduction **5**
- 1.0.1 Location . . . 5
- 1.0.2 End Date . . . 5

2 Overview **6**

3 Contract Base **7**
- 3.1 Constant Definitions . . . 7
- 3.2 Modifier Definitions . . . 8
  - 3.2.1 Modifier signed . . . 8
  - 3.2.2 Modifier accept . . . 8
  - 3.2.3 Modifier onlyContract . . . 9
  - 3.2.4 Modifier onlyMe . . . 9

4 Contract Demiurge **10**
- 4.1 Contract Inheritance . . . 11
- 4.2 Constant Definitions . . . 11
- 4.3 Variable Definitions . . . 11
- 4.4 Modifier Definitions . . . 12
  - 4.4.1 Modifier checksEmpty . . . 12
  - 4.4.2 Modifier onlyStore . . . 12
- 4.5 Constructor Definitions . . . 12
  - 4.5.1 Constructor . . . 12
- 4.6 Public Method Definitions . . . 13
  - 4.6.1 Function deployPadawan . . . 13
  - 4.6.2 Function deployReserveProposal . . . 13
  - 4.6.3 Function getStats . . . 14
  - 4.6.4 Function getStored . . . 14
  - 4.6.5 Function getTotalDistributedCb . . . 15
  - 4.6.6 Function updateAddr . . . 15
  - 4.6.7 Function updateCode . . . 15
- 4.7 Internal Method Definitions . . . 16
  - 4.7.1 Function _allCheckPassed . . . 16
  - 4.7.2 Function _beforeProposalDeploy . . . 16

4.7.3 Function _createChecks . . . 16
4.7.4 Function _deployProposals . . . 17
4.7.5 Function _passCheck . . . 17

5 Contract DemiurgeStore 18
5.1 Overview . . . 18
5.2 General Minor-level Remarks . . . 18
5.3 Public Functions . . . 19
5.3.1 Function queryAddr . . . 19
5.3.2 Function queryCode . . . 19
5.3.3 Function setDensRootAddr . . . 19
5.3.4 Function setFaucetAddr . . . 19
5.3.5 Function setPadawanCode . . . 20
5.3.6 Function setProposalCode . . . 20
5.3.7 Function setTokenRootAddr . . . 20

6 Contract Padawan 21
6.1 Overview . . . 22
6.2 Static Variable Definitions . . . 22
6.3 Variable Definitions . . . 22
6.4 Modifier Definitions . . . 22
6.4.1 Modifier onlyOwner . . . 22
6.4.2 Modifier onlyTokenRoot . . . 23
6.5 Constructor Definitions . . . 23
6.5.1 Constructor . . . 23
6.6 Public Method Definitions . . . 23
6.6.1 Function confirmVote . . . 23
6.6.2 Function depositTokens . . . 24
6.6.3 Function getActiveProposals . . . 24
6.6.4 Function getAddresses . . . 24
6.6.5 Function getAll . . . 24
6.6.6 Function getTipAccount . . . 25
6.6.7 Function getVoteInfo . . . 25
6.6.8 Function onGetBalance . . . 25
6.6.9 Function onTokenWalletDeploy . . . 25
6.6.10 Function reclaimDeposit . . . 25
6.6.11 Function rejectVote . . . 26
6.6.12 Function updateStatus . . . 26
6.6.13 Function vote . . . 27
6.7 Internal Method Definitions . . . 28
6.7.1 Function _createTokenAccount . . . 28
6.7.2 Function _unlockDeposit . . . 28
6.7.3 Function _updateLockedVotes . . . 28

7 Contract PadawanResolver **29**
7.1 Overview . . . 29
7.2 Variable Definitions . . . 29
7.3 Public Method Definitions . . . 29
7.3.1 Function resolvePadawan . . . 29
7.4 Internal Method Definitions . . . 30
7.4.1 Function _buildPadawanState . . . 30

8 Contract Proposal **31**
8.1 Overview . . . 31
8.2 Static Variable Definitions . . . 32
8.3 Variable Definitions . . . 32
8.4 Constructor Definitions . . . 32
8.4.1 Constructor . . . 32
8.5 Public Method Definitions . . . 33
8.5.1 Function getAll . . . 33
8.5.2 Function getCurrentVotes . . . 33
8.5.3 Function getInfo . . . 33
8.5.4 Function getVotingResults . . . 33
8.5.5 Function queryStatus . . . 34
8.5.6 Function vote . . . 34
8.5.7 Function wrapUp . . . 35
8.6 Internal Method Definitions . . . 35
8.6.1 Function _buildPadawanState . . . 35
8.6.2 Function _calculateVotes . . . 35
8.6.3 Function _changeState . . . 35
8.6.4 Function _finalize . . . 36
8.6.5 Function _softMajority . . . 36
8.6.6 Function _tryEarlyComplete . . . 37
8.6.7 Function _wrapUp . . . 37

9 Contract ProposalResolver **38**
9.1 Overview . . . 38
9.2 Variable Definitions . . . 38
9.3 Public Method Definitions . . . 38
9.3.1 Function resolveProposal . . . 38
9.4 Internal Method Definitions . . . 39
9.4.1 Function _buildProposalState . . . 39

Chapter 1

Overview

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

Chapter 2

Contract Base

In file `Base.sol`

2.1 Constant Definitions

```
8     uint16 constant ERROR_DIFFERENT_CALLER =  211;

10    uint64 constant START_BALANCE       = 3 ton;

11    uint64 constant DEPLOYER_FEE        = 0.1 ton;

12    uint64 constant PROCESS_FEE         = 0.3 ton;

13    uint64 constant VOTE_FEE            = 1 ton;

14    uint64 constant DEPLOY_FEE          = START_BALANCE +
          DEPLOYER_FEE;

15    uint64 constant DEPLOY_PAY          = DEPLOY_FEE + PROCESS_FEE;

16    uint64 constant DEPLOY_PROPOSAL_FEE = 5 ton;

17    uint64 constant DEPLOY_PROPOSAL_PAY = DEPLOY_PROPOSAL_FEE +
          PROCESS_FEE;

18    uint64 constant DEPOSIT_TONS_FEE    = 1 ton;

19    uint64 constant DEPOSIT_TONS_PAY    = DEPOSIT_TONS_FEE +
          PROCESS_FEE;

20    uint64 constant DEPOSIT_TOKENS_FEE  = 0.5 ton +
          DEPOSIT_TONS_FEE;

21    uint64 constant DEPOSIT_TOKENS_PAY  = DEPOSIT_TOKENS_FEE +
          PROCESS_FEE;
```

```
22      uint64 constant TOKEN_ACCOUNT_FEE   = 2 ton;

23      uint64 constant TOKEN_ACCOUNT_PAY   = TOKEN_ACCOUNT_FEE +
            PROCESS_FEE;

24      uint64 constant QUERY_STATUS_FEE    = 0.02 ton;

25      uint64 constant QUERY_STATUS_PAY    = QUERY_STATUS_FEE +
            DEF_RESPONSE_VALUE;

27      uint64 constant DEF_RESPONSE_VALUE = 0.03 ton;

28      uint64 constant DEF_COMPUTE_VALUE = 0.2 ton;
```

2.2 Modifier Definitions

2.2.1 Modifier signed

```
30      modifier signed {
31          require(msg.pubkey() == tvm.pubkey(), Errors.INVALID_CALLER
                ());
32          tvm.accept();
33          _;
34      }
```

2.2.2 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.

```
36      modifier accept {
37          tvm.accept();
38          _;
39      }
```

2.2.3 Modifier onlyContract

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

2.2.4 Modifier onlyMe

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

Chapter 3

Contract Demiurge

Contents

3.1 Constant Definitions 7
3.2 Modifier Definitions 8
3.2.1 Modifier signed 8
3.2.2 Modifier accept 8
3.2.3 Modifier onlyContract 9
3.2.4 Modifier onlyMe 9

In file `Demiurge.sol`

The `Demiurge` contract acts as a central hub to create user contracts and proposal contracts.x

3.1 Contract Inheritance

Base	
PadawanResolver	
ProposalResolver	
IDemiurgeStoreCb	
IFaucetCb	

3.2 Constant Definitions

- OK

```
30    uint8 constant CHECK_PROPOSAL = 1;

31    uint8 constant CHECK_PADAWAN = 2;

33    uint128 constant TOTAL_EMISSION = 21000000;
```

3.3 Variable Definitions

- OK

```
35    uint32 _deployedPadawansCounter = 0;

36    uint32 _deployedProposalsCounter = 0;

37    uint16 _version = 3;

39    address _addrStore;

40    address _addrDensRoot;

41    address _addrTokenRoot;

42    address _addrFaucet;

44    uint8 _checkList;

46    NewProposal[] public _newProposals;

47    uint8 public _getBalancePendings = 0;

48    uint128 public _totalVotes = 0;
```

3.4 Modifier Definitions

3.4.1 Modifier checksEmpty

- Minor issue: this modifier is not used. It should be removed.

```
66    modifier checksEmpty() {
67        require(_allCheckPassed(), Errors.NOT_ALL_CHECKS_PASSED);
68        tvm.accept();
69        _;
70    }
```

3.4.2 Modifier onlyStore

- OK

```
72    modifier onlyStore() {
73        require(msg.sender == _addrStore);
74        tvm.accept();
75        _;
76    }
```

3.5 Constructor Definitions

3.5.1 Constructor

- **Critical issue: Administrative Take-over in `Demiurge.constructor`**
  No test is performed to verify the sender in the case `msg.sender != address(0)`. An attacker could use it to deploy the contract himself for another user, providing its own `addrStore`, i.e. with his own code for most contracts.

- **Major issue: No initialization check performed in `Demiurge.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.

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

- Minor issue (duplicate code): the check `addrStore != address(0)` is performed twice, the second one is useless.

```
82      constructor(address addrStore) public {
83          if (msg.sender == address(0)) {
84              require(msg.pubkey() == tvm.pubkey(), 101);
85          }
86          require(addrStore != address(0), Errors.
                STORE_SHOULD_BE_NOT_NULL);
87          tvm.accept();
88
89          if (addrStore != address(0)) {
90              _addrStore = addrStore;
91              DemiurgeStore(_addrStore).queryCode{value: 0.2 ton,
                    bounce: true}(ContractType.Proposal);
92              DemiurgeStore(_addrStore).queryCode{value: 0.2 ton,
                    bounce: true}(ContractType.Padawan);
93              DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
                    bounce: true}(ContractAddr.DensRoot);
94              DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
                    bounce: true}(ContractAddr.TokenRoot);
95              DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
                    bounce: true}(ContractAddr.Faucet);
96          }
97
98          _createChecks();
99      }
```

3.6 Public Method Definitions

3.6.1 Function deployPadawan

- Minor issue: the function should check that the code of the Padawan contract was correctly initialized.

```
103     function deployPadawan(address owner) external onlyContract {
104         require(msg.value >= DEPLOY_FEE + 2 ton);
105         require(owner != address(0));
106         TvmCell state = _buildPadawanState(owner);
107         new Padawan{stateInit: state, value: START_BALANCE + 2 ton
                }(_addrTokenRoot);
108     }
```

3.6.2 Function deployReserveProposal

- Minor issue: this function should check that `_codePadawan` and `_codeProposal` have been correctly initialized
- Minor issue: there is no need to store `_codePadawan` in the proposal struct as it is already a global variable.

```
112     function deployReserveProposal(
113         string title,
114         ReserveProposalSpecific specific
115     ) external onlyContract {
116         require(msg.value >= DEPLOY_PROPOSAL_FEE);
117         TvmBuilder b;
118         b.store(specific);
119         TvmCell cellSpecific = b.toCell();
120 
121         NewProposal _newProposal = NewProposal(
122             0,
123             _addrDensRoot,
124             ProposalType.Reserve,
125             cellSpecific,
126             _codePadawan,
127             _buildProposalState(title)
128         );
129         _newProposals.push(_newProposal);
130 
131         _beforeProposalDeploy(uint8(_newProposals.length - 1));
132     }
```

3.6.3 Function getStats

- OK

```
214     function getStats() public view returns (uint16 version, uint32
            deployedPadawansCounter, uint32 deployedProposalsCounter)
        {
215         version = _version;
216         deployedPadawansCounter = _deployedPadawansCounter;
217         deployedProposalsCounter = _deployedProposalsCounter;
218     }
```

3.6.4 Function getStored

- OK

```
198     function getStored() public view returns (
199         TvmCell codePadawan,
200         TvmCell codeProposal,
201         address addrStore,
202         address addrDensRoot,
203         address addrTokenRoot,
204         address addrFaucet
205     ) {
206         codePadawan = _codePadawan;
207         codeProposal = _codeProposal;
208         addrStore = _addrStore;
209         addrDensRoot = _addrDensRoot;
210         addrTokenRoot = _addrTokenRoot;
211         addrFaucet = _addrFaucet;
212     }
```

3.6.5 Function getTotalDistributedCb

- **Critical issue: No permission check in `Demiurge.getTotalDistributedCb`**
  Anybody can send this message. An attacker could use it to force the deployment of all proposals with a wrong number of total votes.
- **Critical issue: No value check in `Demiurge.getTotalDistributedCb`**
  This function is in charge of deploying all pending proposals. It should check that the sender gave enough value to perform these deployments before the end of the action phase. Otherwise, the action phase may succeed, all proposal will be removed from the array of proposals, but the deployments will fail by lack of gas.
- Minor issue: this function should send back the remaining gas not consumed to its called, especially if the caller gave a lot of gas to account for the deployments of multiple proposals.

```
148     function getTotalDistributedCb(
149         uint128 totalDistributed
150     ) public override {
151         _totalVotes = totalDistributed;
```

```
152         _getBalancePendings -= 1;
153         _deployProposals();
154     }
```

3.6.6 Function updateAddr

- Minor issue: add `_passCheck` for addresses too.

```
174     function updateAddr(ContractAddr kind, address addr) external
            override onlyStore {
175         require(addr != address(0));
176         if (kind == ContractAddr.DensRoot) {
177             _addrDensRoot = addr;
178         } else if (kind == ContractAddr.TokenRoot) {
179             _addrTokenRoot = addr;
180         } else if (kind == ContractAddr.Faucet) {
181             _addrFaucet = addr;
182         }
183     }
```

3.6.7 Function updateCode

- OK

```
185     function updateCode(ContractType kind, TvmCell code) external
            override onlyStore {
186         tvm.accept();
187         if (kind == ContractType.Proposal) {
188             _codeProposal = code;
189             _passCheck(CHECK_PROPOSAL);
190         } else if (kind == ContractType.Padawan) {
191             _codePadawan = code;
192             _passCheck(CHECK_PADAWAN);
193         }
194     }
```

3.7 Internal Method Definitions

3.7.1 Function _allCheckPassed

- OK

```
62     function _allCheckPassed() private view inline returns (bool) {
63         return (_checkList == 0);
64     }
```

3.7.2 Function _beforeProposalDeploy

- OK

```
134     function _beforeProposalDeploy(
135         uint8 i
136     ) private {
137         uint256 hashState = tvm.hash(_newProposals[i].state);
138         address addrProposal = address.makeAddrStd(0, hashState);
139         IClient(_addrDensRoot).onProposalDeploy
140             {value: 1 ton, bounce: true}
141             (addrProposal, _newProposals[i].proposalType,
                    _newProposals[i].specific);
142 
143         IFaucet(_addrFaucet).getTotalDistributed
144             {value: 0.2 ton, flag: 1, bounce: false}();
145         _getBalancePendings += 1;
146     }
```

3.7.3 Function _createChecks

- OK

```
54      function _createChecks() private inline {
55          _checkList = CHECK_PADAWAN | CHECK_PROPOSAL;
56      }
```

3.7.4 Function _deployProposals

- OK

```
156     function _deployProposals() private {
157         if(_getBalancePendings == 0) {
158             for(uint8 i = 0; i < _newProposals.length; i++) {
159                 new Proposal {stateInit: _newProposals[i].state,
                        value: START_BALANCE}(
160                     _totalVotes,
161                     _newProposals[i].addrClient,
162                     _newProposals[i].proposalType,
163                     _newProposals[i].specific,
164                     _newProposals[i].codePadawan
165                 );
166                 _deployedProposalsCounter++;
167             }
168             delete _newProposals;
169         }
170     }
```

3.7.5 Function _passCheck

- OK

```
58      function _passCheck(uint8 check) private inline {
59          _checkList &= ~check;
60      }
```

Chapter 4

Contract DemiurgeStore

Contents

4.1 Contract Inheritance 11
4.2 Constant Definitions 11
4.3 Variable Definitions 11
4.4 Modifier Definitions 12
4.4.1 Modifier checksEmpty 12
4.4.2 Modifier onlyStore 12
4.5 Constructor Definitions 12
4.5.1 Constructor 12
4.6 Public Method Definitions 13
4.6.1 Function deployPadawan 13
4.6.2 Function deployReserveProposal 13
4.6.3 Function getStats 14
4.6.4 Function getStored 14
4.6.5 Function getTotalDistributedCb 15
4.6.6 Function updateAddr 15
4.6.7 Function updateCode 15
4.7 Internal Method Definitions 16
4.7.1 Function _allCheckPassed 16
4.7.2 Function _beforeProposalDeploy 16
4.7.3 Function _createChecks 16
4.7.4 Function _deployProposals 17
4.7.5 Function _passCheck 17

4.1 Overview

In file `DemiurgeStore.sol`

This contract is used to store "global" values for the whole infrastructure, such as the code of the contracts to be deployed and the addresses of some contracts.

4.2 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).

4.3 Public Functions

4.3.1 Function queryAddr

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

```
43      function queryAddr(ContractAddr kind) public view {
44          address addr = _addrs[uint8(kind)];
45          IDemiurgeStoreCb(msg.sender).updateAddr{value: 0, flag: 64,
                bounce: false}(kind, addr);
46      }
```

4.3.2 Function queryCode

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

```
38      function queryCode(ContractType kind) public view {
39          TvmCell code = _codes[uint8(kind)];
40          IDemiurgeStoreCb(msg.sender).updateCode{value: 0, flag: 64,
                bounce: false}(kind, code);
41      }
```

4.3.3 Function setDensRootAddr

- OK

```
21      function setDensRootAddr(address addr) public signed {
22          require(addr != address(0));
23          _addrs[uint8(ContractAddr.DensRoot)] = addr;
24      }
```

4.3.4 Function setFaucetAddr

- OK

```
29      function setFaucetAddr(address addr) public signed {
30          require(addr != address(0));
31          _addrs[uint8(ContractAddr.Faucet)] = addr;
32      }
```

4.3.5 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`

```
14      function setPadawanCode(TvmCell code) public signed {
15          _codes[uint8(ContractType.Padawan)] = code;
16      }
```

4.3.6 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`

```
17      function setProposalCode(TvmCell code) public signed {
18          _codes[uint8(ContractType.Proposal)] = code;
19      }
```

4.3.7 Function setTokenRootAddr

- OK

```
25      function setTokenRootAddr(address addr) public signed {
26          require(addr != address(0));
27          _addrs[uint8(ContractAddr.TokenRoot)] = addr;
28      }
```

Chapter 5

Contract Padawan

Contents

5.1 Overview 18
5.2 General Minor-level Remarks 18
5.3 Public Functions 19
5.3.1 Function queryAddr 19
5.3.2 Function queryCode 19
5.3.3 Function setDensRootAddr 19
5.3.4 Function setFaucetAddr 19
5.3.5 Function setPadawanCode 20
5.3.6 Function setProposalCode 20
5.3.7 Function setTokenRootAddr 20

5.1 Overview

In file `Padawan.sol`

This contract is used by a user to collect his voting rights (within a token wallet), and vote for proposals. Voting rights can be added, and reclaimed if not currently used.

5.2 Static Variable Definitions

- OK

```
18    address static _deployer;

19    address static _owner;
```

5.3 Variable Definitions

- Minor issue: there is no function to clean `_activeProposals`, i.e. to remove proposals that are ended. Currently, it is possible to use `reclaimDeposit` with argument 0 to do that. It would be better to introduce a `cleanProposals` function for that purpose.

```
21     address _addrTokenRoot;

23     TipAccount _tipAccount;

24     address _returnTo;

26     mapping(address => uint32) _activeProposals;

28     uint32 _requestedVotes;

29     uint32 _totalVotes;

30     uint32 _lockedVotes;
```

5.4 Modifier Definitions

5.4.1 Modifier onlyOwner

- OK

```
34     modifier onlyOwner() {
35         require(msg.sender == _owner, Errors.
               NOT_AUTHORIZED_CONTRACT);
36         _;
37     }
```

5.4.2 Modifier onlyTokenRoot

- OK

```
39     modifier onlyTokenRoot() {
40         require(msg.sender == _addrTokenRoot, Errors.INVALID_CALLER
               );
41         _;
42     }
```

5.5 Constructor Definitions

5.5.1 Constructor

- OK

```
46      constructor(address addrTokenRoot) public onlyContract {
47          require(_deployer == msg.sender, Errors.ONLY_DEPLOYER);
48          _addrTokenRoot = addrTokenRoot;
49          _createTokenAccount();
50      }
```

5.6 Public Method Definitions

5.6.1 Function confirmVote

- Minor issue: there is no real reason to call `_updateLockedVotes` here, as it could be called in `reclaimDeposit` instead. Indeed, `_lockedVotes` is only used when the deposit is reclaimed, so it will save the cost of the recomputation if the user votes for many proposals without reclaiming his tokens.

```
74      function confirmVote(uint32 votesCount) external onlyContract {
75          // TODO: better to check is it proposal or not
76          optional(uint32) optActiveProposal = _activeProposals.fetch
                (msg.sender);
77          require(optActiveProposal.hasValue());
78
79          _activeProposals[msg.sender] += votesCount;
80
81          _updateLockedVotes();
82
83          _owner.transfer(0, false, 64);
84      }
```

5.6.2 Function depositTokens

- OK

```
172     function depositTokens() external onlyOwner view {
173         require(msg.value >= DEPOSIT_TOKENS_FEE, Errors.
                MSG_VALUE_TOO_LOW);
174         require(_tipAccount.addr != address(0), Errors.
                ACCOUNT_DOES_NOT_EXIST);
175
176         ITokenWallet(_tipAccount.addr).getBalance_InternalOwner
177             {value: 0, flag: 64, bounce: true}
178             (tvm.functionId(onGetBalance));
179     }
```

5.6.3 Function getActiveProposals

- OK

```
228     function getActiveProposals() public view returns (mapping(
            address => uint32) activeProposals) {
229         activeProposals = _activeProposals;
230     }
```

5.6.4 Function getAddresses

- OK

```
224     function getAddresses() public view returns (address
            ownerAddress) {
225         ownerAddress = _owner;
226     }
```

5.6.5 Function getAll

- OK

```
207     function getAll() external view returns (TipAccount tipAccount,
             uint32 reqVotes, uint32 totalVotes, uint32 lockedVotes) {
208         tipAccount = _tipAccount;
209         reqVotes = _requestedVotes;
210         totalVotes =  _totalVotes;
211         lockedVotes = _lockedVotes;
212     }
```

5.6.6 Function getTipAccount

- OK

```
214     function getTipAccount() external view returns (TipAccount
            tipAccount) {
215         tipAccount = _tipAccount;
216     }
```

5.6.7 Function getVoteInfo

- OK

```
218     function getVoteInfo() external view returns (uint32 reqVotes,
            uint32 totalVotes, uint32 lockedVotes) {
219         reqVotes = _requestedVotes;
220         totalVotes =  _totalVotes;
221         lockedVotes = _lockedVotes;
222     }
```

5.6.8 Function onGetBalance

- OK

```
181     function onGetBalance(uint128 balance) public onlyContract {
182         require(_tipAccount.addr == msg.sender, Errors.
                NOT_AUTHORIZED_CONTRACT);
183         _tipAccount.balance = balance;
184         _totalVotes = uint32(balance);
185         _owner.transfer(0, false, 64);
186     }
```

5.6.9 Function onTokenWalletDeploy

- OK

```
192     function onTokenWalletDeploy(address ownerAddress) public
            onlyTokenRoot {
193         _tipAccount = TipAccount(ownerAddress, 0);
194         _owner.transfer(0, false, 64);
195     }
```

5.6.10 Function reclaimDeposit

- Minor issue: the user might want to use `votes=0` to cancel a withdrawal. In this case, this function should skip sending all `queryStatus` messages, unless the goal is to clean the `_activeProposals` mapping (we advise to create a function for that purpose).
- Minor issue: there is no reason to send `queryStatus` messages if the `_unlockDeposit` function was called, i.e. if the reclaim was already successful

```
103     function reclaimDeposit(uint32 votes, address returnTo)
            external onlyOwner {
104         require(msg.value >= 3 ton, Errors.MSG_VALUE_TOO_LOW);
105         require(votes <= _totalVotes, Errors.NOT_ENOUGH_VOTES);
106         require(returnTo != address(0));
107         _returnTo = returnTo;
108         _requestedVotes = votes;
109
110         if (_requestedVotes <= _totalVotes - _lockedVotes) {
111             _unlockDeposit();
112         } else {
113             _requestedVotes = 0;
114         }
115
116         optional(address, uint32) optActiveProposal =
                _activeProposals.min();
117         while (optActiveProposal.hasValue()) {
118             (address addrActiveProposal,) = optActiveProposal.get()
                    ;
```

```
119             IProposal(addrActiveProposal).queryStatus
120                 {value: QUERY_STATUS_FEE, bounce: true, flag: 1}
121                 ();
122             optActiveProposal = _activeProposals.next(
                    addrActiveProposal);
123         }
124     }
```

5.6.11 Function rejectVote

- OK

```
87      function rejectVote(uint32 votesCount, uint16 errorCode)
            external onlyContract {
88          votesCount; errorCode;
89
90          // TODO: better to check is it proposal or not
91          optional(uint32) optActiveProposal = _activeProposals.fetch
                (msg.sender);
92          require(optActiveProposal.hasValue());
93          uint32 activeProposalVotes = optActiveProposal.get();
94          if (activeProposalVotes == 0) {
95              delete _activeProposals[msg.sender];
96          }
97
98          _owner.transfer(0, false, 64);
99      }
```

5.6.12 Function updateStatus

- OK

```
127     function updateStatus(ProposalState state) external
            onlyContract {
128         optional(uint32) optActiveProposal = _activeProposals.fetch
                (msg.sender);
129         require(optActiveProposal.hasValue());
130         tvm.accept();
131
132         if (state >= ProposalState.Ended) {
133             delete _activeProposals[msg.sender];
134             _updateLockedVotes();
135         }
136
137         if (_requestedVotes != 0 && _requestedVotes <= _totalVotes
                - _lockedVotes) {
138             _unlockDeposit();
139         }
140     }
```

5.6.13 Function vote

- **Critical issue: Unlimited voting rights in `Padawan.vote`**
  An attacker can call this method several times in the same round and in consecutive rounds to vote several times for the same proposal, until the `Padawan.confirmVote` message is received. Fix: voting rights should be immediately decreased instead of waiting for `confirmVote`.

- **Major issue: Infinite locking of deposits in `Padawan.vote`**
  An attacker could send a faked proposal address to a user to make him vote for a non-existing proposal. It can generate a little increase in storage, but if the fix of the critical issue above is done, it could also lock the deposits forever, as the corresponding contract will never end and unlock the deposits. Fix: this method should take the title of the proposal in argument, computes the address of the proposal, and the contract should correctly deal with bounced messages.

```
55      function vote(address proposal, bool choice, uint32 votes)
            external onlyOwner {
56          require(msg.value >= VOTE_FEE, Errors.MSG_VALUE_TOO_LOW);
57          optional(uint32) optActiveProposal = _activeProposals.fetch
                (proposal);
58
59          uint32 activeProposalVotes = optActiveProposal.hasValue() ?
                 optActiveProposal.get() : 0;
60          uint32 availableVotes = _totalVotes - activeProposalVotes;
61          require(votes <= availableVotes, Errors.NOT_ENOUGH_VOTES);
62
63          // TODO: better to remove
64          if (activeProposalVotes == 0) {
65              _activeProposals[proposal] = 0;
66          }
67
68          IProposal(proposal).vote
69              {value: 0, flag: 64, bounce: true}
70              (_owner, choice, votes);
71      }
```

5.7 Internal Method Definitions

5.7.1 Function _createTokenAccount

- OK

```
197     function _createTokenAccount() private view {
198         ITokenRoot(_addrTokenRoot).deployEmptyWallet
199             {value: 2 ton, flag: 1, bounce: true}
200             (tvm.functionId(onTokenWalletDeploy), 0, 0, address(
                    this).value, 1 ton);
201     }
```

5.7.2 Function _unlockDeposit

- Minor issue: this function should skip sending a message if `_requestedVotes` is 0.

```
146     function _unlockDeposit() private {
147         ITokenWallet(_tipAccount.addr).transfer
148             {value: 0.1 ton + 0.1 ton}
149             (_returnTo, _requestedVotes, 0.1 ton);
150         _totalVotes -= _requestedVotes;
151         _requestedVotes = 0;
152         _returnTo = address(0);
153     }
```

5.7.3 Function _updateLockedVotes

- OK

```
155     function _updateLockedVotes() private inline {
156         optional(address, uint32) optActiveProposal =
                _activeProposals.min();
157         uint32 lockedVotes;
158         while (optActiveProposal.hasValue()) {
159             (address addr, uint32 votes) = optActiveProposal.get();
160             if (votes > lockedVotes) {
161                 lockedVotes = votes;
162             }
163             optActiveProposal = _activeProposals.next(addr);
164         }
165         _lockedVotes = lockedVotes;
166     }
```

Chapter 6

Contract PadawanResolver

Contents

6.1 Overview 22
6.2 Static Variable Definitions 22
6.3 Variable Definitions 22
6.4 Modifier Definitions 22
6.4.1 Modifier onlyOwner 22
6.4.2 Modifier onlyTokenRoot 23
6.5 Constructor Definitions 23
6.5.1 Constructor 23
6.6 Public Method Definitions 23
6.6.1 Function confirmVote 23
6.6.2 Function depositTokens 24
6.6.3 Function getActiveProposals 24
6.6.4 Function getAddresses 24
6.6.5 Function getAll 24
6.6.6 Function getTipAccount 25
6.6.7 Function getVoteInfo 25
6.6.8 Function onGetBalance 25
6.6.9 Function onTokenWalletDeploy 25
6.6.10 Function reclaimDeposit 25
6.6.11 Function rejectVote 26
6.6.12 Function updateStatus 26
6.6.13 Function vote 27
6.7 Internal Method Definitions 28
6.7.1 Function _createTokenAccount 28
6.7.2 Function _unlockDeposit 28
6.7.3 Function _updateLockedVotes 28

6.1 Overview

In file `PadawanResolver.sol`

This contract is inherited by contracts that need to deploy `Padawan` contract and verify that an address belongs to a deployed `Padawan` contract.

6.2 Variable Definitions

- OK

```
8     TvmCell _codePadawan;
```

6.3 Public Method Definitions

6.3.1 Function resolvePadawan

- OK

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

6.4 Internal Method Definitions

6.4.1 Function _buildPadawanState

- Minor issue: the state built in this function uses `address(this)` as one of the static variables for the contract. Yet, this contract is bound to be inherited by different contracts (here, at least `Demiurge` and `Proposal`), i.e. computed addresses will be different for different contracts. Instead, the value of the `_deployer` variable should be made explicit to the caller, by passing it as an argument of the function.
- Minor issue: this function should fail (`require`) if the `_codePadawan` variable has not yet been initialized. A global boolean could be used for that, set in an internal function initializing both global variables.

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

```
20              code: _codePadawan
21          });
22      }
```

Chapter 7

Contract Proposal

Contents

7.1 Overview 29
7.2 Variable Definitions 29
7.3 Public Method Definitions 29
7.3.1 Function resolvePadawan 29
7.4 Internal Method Definitions 30
7.4.1 Function _buildPadawanState 30

7.1 Overview

In file `Proposal.sol`

This contract is used to collect the votes for a particular proposal. Votes are sent by `Padawan` contracts.

7.2 Static Variable Definitions

- OK

```
13    address static _deployer;
```

```
14    string static _title;
```

7.3 Variable Definitions

- OK

```
16    address public _addrClient;
```

```
18      ProposalInfo public _proposalInfo;

20      ProposalResults _results;

21      VoteCountModel _voteCountModel;
```

7.4 Constructor Definitions

7.4.1 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.

```
25      constructor(
26          uint128 totalVotes,
27          address addrClient,
28          ProposalType proposalType,
29          TvmCell specific,
30          TvmCell codePadawan
31      ) public {
32          require(_deployer == msg.sender);
33
34          _addrClient = addrClient;
35
36          _proposalInfo.title = _title;
37          _proposalInfo.start = uint32(now);
38          _proposalInfo.end = uint32(now + 60 * 60 * 24 * 7);
39          _proposalInfo.proposalType = proposalType;
40          _proposalInfo.specific = specific;
41          _proposalInfo.state = ProposalState.New;
42          _proposalInfo.totalVotes = totalVotes;
43
44          _codePadawan = codePadawan;
45
46          _voteCountModel = VoteCountModel.SoftMajority;
47      }
```

7.5 Public Method Definitions

7.5.1 Function getAll

- OK

```
168     function getAll() public view override returns (ProposalInfo
            info) {
169         info = _proposalInfo;
170     }
```

7.5.2 Function getCurrentVotes

- OK

```
181     function getCurrentVotes() external override view returns (
            uint32 votesFor, uint32 votesAgainst) {
182         return (_proposalInfo.votesFor, _proposalInfo.votesAgainst)
                ;
183     }
```

7.5.3 Function getInfo

- OK

```
177     function getInfo() public view returns (ProposalInfo info) {
178         info = _proposalInfo;
179     }
```

7.5.4 Function getVotingResults

- OK

```
172     function getVotingResults() public view returns (
            ProposalResults vr) {
173         require(_proposalInfo.state > ProposalState.Ended, Errors.
                VOTING_HAS_NOT_ENDED);
174         vr = _results;
175     }
```

7.5.5 Function queryStatus

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

```
162     function queryStatus() external override {
163         IPadawan(msg.sender).updateStatus(_proposalInfo.state);
164     }
```

7.5.6 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.

```
55      function vote(address addrPadawanOwner, bool choice, uint32
            votesCount) external override {
56          address addrPadawan = resolvePadawan(addrPadawanOwner);
57          uint16 errorCode = 0;
58
59          if (addrPadawan != msg.sender) {
60              errorCode = Errors.NOT_AUTHORIZED_CONTRACT;
61          } else if (now < _proposalInfo.start) {
62              errorCode = Errors.VOTING_NOT_STARTED;
63          } else if (now > _proposalInfo.end) {
64              errorCode = Errors.VOTING_HAS_ENDED;
65          }
66
67          if (errorCode > 0) {
68              IPadawan(msg.sender).rejectVote{value: 0, flag: 64,
                    bounce: true}(votesCount, errorCode);
69          } else {
70              IPadawan(msg.sender).confirmVote{value: 0, flag: 64,
                    bounce: true}(votesCount);
71              if (choice) {
72                  _proposalInfo.votesFor += votesCount;
73              } else {
74                  _proposalInfo.votesAgainst += votesCount;
75              }
76          }
77
78          _wrapUp();
79      }
```

7.5.7 Function wrapUp

- OK

```
49      function wrapUp() external override {
50          _wrapUp();
51          msg.sender.transfer(0, false, 64);
52      }
```

7.6 Internal Method Definitions

7.6.1 Function _buildPadawanState

- Minor issue (code repetition): instead of defining this function, the same function in `PadawanResolver` should take the `deployer` in argument.

```
154     function _buildPadawanState(address owner) internal view
            override returns (TvmCell) {
155         return tvm.buildStateInit({
156             contr: Padawan,
157             varInit: {_deployer: _deployer, _owner: owner},
158             code: _codePadawan
159         });
160     }
```

7.6.2 Function _calculateVotes

- OK

```
132     function _calculateVotes(
133         uint32 yes,
134         uint32 no
135     ) private view returns (bool) {
136         bool passed = false;
137         passed = _softMajority(yes, no);
138         return passed;
139     }
```

7.6.3 Function _changeState

- OK

```
150     function _changeState(ProposalState state) private inline {
151         _proposalInfo.state = state;
152     }
```

7.6.4 Function _finalize

- Minor issue: a `require` should check that the message contains enough value to send the `onProposalPassed` message. This check could be moved earlier in methods calling `_finalize`

```
81      function _finalize(bool passed) private {
82          _results = ProposalResults(
83              uint32(0),
84              passed,
85              _proposalInfo.votesFor,
86              _proposalInfo.votesAgainst,
```

```
87              _proposalInfo.totalVotes,
88              _voteCountModel,
89              uint32(now)
90          );
91
92          ProposalState state = passed ? ProposalState.Passed :
                ProposalState.NotPassed;
93
94          _changeState(state);
95
96          IClient(address(_addrClient)).onProposalPassed{value: 1 ton
                } (_proposalInfo);
97
98          emit ProposalFinalized(_results);
99      }
```

7.6.5 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.

```
141     function _softMajority(
142         uint32 yes,
143         uint32 no
144     ) private view returns (bool) {
145         bool passed = false;
146         passed = yes >= 1 + (_proposalInfo.totalVotes / 10) + (no *
                ((_proposalInfo.totalVotes / 2) - (_proposalInfo.
                totalVotes / 10))) / (_proposalInfo.totalVotes / 2);
147         return passed;
148     }
```

7.6.6 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.

```
101     function _tryEarlyComplete(
102         uint32 yes,
103         uint32 no
104     ) private view returns (bool, bool) {
105         (bool completed, bool passed) = (false, false);
```

```
106         if (yes * 2 > _proposalInfo.totalVotes) {
107             completed = true;
108             passed = true;
109         } else if(no * 2 >= _proposalInfo.totalVotes) {
110             completed = true;
111             passed = false;
112         }
113         return (completed, passed);
114     }
```

7.6.7 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;

```
116     function _wrapUp() private {
117         (bool completed, bool passed) = (false, false);
118 
119         if (now > _proposalInfo.end) {
120             completed = true;
121             passed = _calculateVotes(_proposalInfo.votesFor,
                    _proposalInfo.votesAgainst);
122         } else {
123             (completed, passed) = _tryEarlyComplete(_proposalInfo.
                    votesFor, _proposalInfo.votesAgainst);
124         }
125 
126         if (completed) {
127             _changeState(ProposalState.Ended);
128             _finalize(passed);
129         }
130     }
```

Chapter 8

Contract ProposalResolver

Contents

8.1 Overview 31
8.2 Static Variable Definitions 32
8.3 Variable Definitions 32
8.4 Constructor Definitions 32
8.4.1 Constructor 32
8.5 Public Method Definitions 33
8.5.1 Function getAll 33
8.5.2 Function getCurrentVotes 33
8.5.3 Function getInfo 33
8.5.4 Function getVotingResults 33
8.5.5 Function queryStatus 34
8.5.6 Function vote 34
8.5.7 Function wrapUp 35
8.6 Internal Method Definitions 35
8.6.1 Function _buildPadawanState 35
8.6.2 Function _calculateVotes 35
8.6.3 Function _changeState 35
8.6.4 Function _finalize 36
8.6.5 Function _softMajority 36
8.6.6 Function _tryEarlyComplete 37
8.6.7 Function _wrapUp 37

8.1 Overview

In file `ProposalResolver.sol`

This contract is inherited by contracts that need to deploy `Proposal` contract and verify that an address belongs to a deployed `Proposal` contract.

8.2 Variable Definitions

- OK

```
6     TvmCell _codeProposal;
```

8.3 Public Method Definitions

8.3.1 Function resolveProposal

- OK

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

8.4 Internal Method Definitions

8.4.1 Function _buildProposalState

- Minor issue: the state built in this function uses `address(this)` as one of the static variables for the contract. Yet, this contract is bound to be inherited by different contracts (although here, onlye `Demiurge` uses it), i.e. computed addresses will be different for different contracts. Instead, the value of the `_deployer` variable should be made explicit to the caller, by passing it as an argument of the function.
- Minor issue: this function should fail (`require`) if the `_codeProposal` variable has not yet been initialized. A global boolean could be used for that, set in an internal function initializing both global variables.

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