

Audit

By OCamlPro

August 17, 2021

# Table of Major and Critical Issues

Critical issue: Administrative Take-over in <code>Demiurge.constructor</code>	13
Major issue: No initialization check performed in <code>Demiurge.constructor</code>	13
Critical issue: No permission check in <code>Demiurge.getTotalDistributedCb</code>	16
Critical issue: No value check in <code>Demiurge.getTotalDistributedCb</code>	16
Critical issue: Unlimited voting rights in <code>Padawan.vote</code>	28
Major issue: Infinite locking of deposits in <code>Padawan.vote</code>	28
Critical issue: Division by 0 in <code>Proposal._softMajority</code>	37
Major issue: Overflow in <code>Proposal._tryEarlyComplete</code>	38

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Overview</b>	<b>7</b>
<b>3</b>	<b>Contract Base</b>	<b>8</b>
3.1	Constant Definitions . . . . .	8
3.2	Modifier Definitions . . . . .	9
3.2.1	Modifier signed . . . . .	9
3.2.2	Modifier accept . . . . .	9
3.2.3	Modifier onlyContract . . . . .	10
3.2.4	Modifier onlyMe . . . . .	10
<b>4</b>	<b>Contract Demiurge</b>	<b>11</b>
4.1	Contract Inheritance . . . . .	12
4.2	Constant Definitions . . . . .	12
4.3	Variable Definitions . . . . .	12
4.4	Modifier Definitions . . . . .	13
4.4.1	Modifier checksEmpty . . . . .	13
4.4.2	Modifier onlyStore . . . . .	13
4.5	Constructor Definitions . . . . .	13
4.5.1	Constructor . . . . .	13
4.6	Public Method Definitions . . . . .	14
4.6.1	Function deployPadawan . . . . .	14
4.6.2	Function deployReserveProposal . . . . .	14
4.6.3	Function getStats . . . . .	15
4.6.4	Function getStored . . . . .	15
4.6.5	Function getTotalDistributedCb . . . . .	16
4.6.6	Function updateAddr . . . . .	16
4.6.7	Function updateCode . . . . .	16
4.7	Internal Method Definitions . . . . .	17
4.7.1	Function _allCheckPassed . . . . .	17
4.7.2	Function _beforeProposalDeploy . . . . .	17
4.7.3	Function _createChecks . . . . .	17
4.7.4	Function _deployProposals . . . . .	18

4.7.5	Function <code>_passCheck</code> . . . . .	18
<b>5</b>	<b>Contract <code>DemiurgeStore</code></b>	<b>19</b>
5.1	Overview . . . . .	19
5.2	General Minor-level Remarks . . . . .	19
5.3	Public Functions . . . . .	20
5.3.1	Function <code>queryAddr</code> . . . . .	20
5.3.2	Function <code>queryCode</code> . . . . .	20
5.3.3	Function <code>setDensRootAddr</code> . . . . .	20
5.3.4	Function <code>setFaucetAddr</code> . . . . .	20
5.3.5	Function <code>setPadawanCode</code> . . . . .	21
5.3.6	Function <code>setProposalCode</code> . . . . .	21
5.3.7	Function <code>setTokenRootAddr</code> . . . . .	21
<b>6</b>	<b>Contract <code>Padawan</code></b>	<b>22</b>
6.1	Overview . . . . .	23
6.2	Static Variable Definitions . . . . .	23
6.3	Variable Definitions . . . . .	23
6.4	Modifier Definitions . . . . .	23
6.4.1	Modifier <code>onlyOwner</code> . . . . .	23
6.4.2	Modifier <code>onlyTokenRoot</code> . . . . .	24
6.5	Constructor Definitions . . . . .	24
6.5.1	Constructor . . . . .	24
6.6	Public Method Definitions . . . . .	24
6.6.1	Function <code>confirmVote</code> . . . . .	24
6.6.2	Function <code>depositTokens</code> . . . . .	25
6.6.3	Function <code>getActiveProposals</code> . . . . .	25
6.6.4	Function <code>getAddresses</code> . . . . .	25
6.6.5	Function <code>getAll</code> . . . . .	25
6.6.6	Function <code>getTipAccount</code> . . . . .	26
6.6.7	Function <code>getVoteInfo</code> . . . . .	26
6.6.8	Function <code>onGetBalance</code> . . . . .	26
6.6.9	Function <code>onTokenWalletDeploy</code> . . . . .	26
6.6.10	Function <code>reclaimDeposit</code> . . . . .	26
6.6.11	Function <code>rejectVote</code> . . . . .	27
6.6.12	Function <code>updateStatus</code> . . . . .	27
6.6.13	Function <code>vote</code> . . . . .	28
6.7	Internal Method Definitions . . . . .	29
6.7.1	Function <code>_createTokenAccount</code> . . . . .	29
6.7.2	Function <code>_unlockDeposit</code> . . . . .	29
6.7.3	Function <code>_updateLockedVotes</code> . . . . .	29

<b>7</b>	<b>Contract PadawanResolver</b>	<b>30</b>
7.1	Overview . . . . .	30
7.2	Variable Definitions . . . . .	30
7.3	Public Method Definitions . . . . .	30
7.3.1	Function resolvePadawan . . . . .	30
7.4	Internal Method Definitions . . . . .	31
7.4.1	Function _buildPadawanState . . . . .	31
<b>8</b>	<b>Contract Proposal</b>	<b>32</b>
8.1	Overview . . . . .	32
8.2	Static Variable Definitions . . . . .	33
8.3	Variable Definitions . . . . .	33
8.4	Constructor Definitions . . . . .	33
8.4.1	Constructor . . . . .	33
8.5	Public Method Definitions . . . . .	34
8.5.1	Function getAll . . . . .	34
8.5.2	Function getCurrentVotes . . . . .	34
8.5.3	Function getInfo . . . . .	34
8.5.4	Function getVotingResults . . . . .	34
8.5.5	Function queryStatus . . . . .	35
8.5.6	Function vote . . . . .	35
8.5.7	Function wrapUp . . . . .	36
8.6	Internal Method Definitions . . . . .	36
8.6.1	Function _buildPadawanState . . . . .	36
8.6.2	Function _calculateVotes . . . . .	36
8.6.3	Function _changeState . . . . .	36
8.6.4	Function _finalize . . . . .	37
8.6.5	Function _softMajority . . . . .	37
8.6.6	Function _tryEarlyComplete . . . . .	38
8.6.7	Function _wrapUp . . . . .	38
<b>9</b>	<b>Contract ProposalResolver</b>	<b>39</b>
9.1	Overview . . . . .	39
9.2	Variable Definitions . . . . .	39
9.3	Public Method Definitions . . . . .	39
9.3.1	Function resolveProposal . . . . .	39
9.4	Internal Method Definitions . . . . .	40
9.4.1	Function _buildProposalState . . . . .	40

# To edit this document

In the `report.tex` file, choose:

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

Issues can be entered with:

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

## Chapter 1

# Introduction

## Chapter 2

# Overview



## Chapter 3

# Contract Base

### Contents

<b>3.1</b>	<b>Constant Definitions</b>	<b>8</b>
<b>3.2</b>	<b>Modifier Definitions</b>	<b>9</b>
3.2.1	Modifier signed	9
3.2.2	Modifier accept	9
3.2.3	Modifier onlyContract	10
3.2.4	Modifier onlyMe	10

In file `Base.sol`

### 3.1 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;

```

## 3.2 Modifier Definitions

### 3.2.1 Modifier signed

```

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

```

### 3.2.2 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      tvvm.accept();
38      -;
39  }

```

### 3.2.3 Modifier onlyContract

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

### 3.2.4 Modifier onlyMe

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

## Chapter 4

# Contract Demiurge

### Contents

---

<b>4.1</b>	<b>Contract Inheritance</b>	<b>12</b>
<b>4.2</b>	<b>Constant Definitions</b>	<b>12</b>
<b>4.3</b>	<b>Variable Definitions</b>	<b>12</b>
<b>4.4</b>	<b>Modifier Definitions</b>	<b>13</b>
4.4.1	Modifier checksEmpty	13
4.4.2	Modifier onlyStore	13
<b>4.5</b>	<b>Constructor Definitions</b>	<b>13</b>
4.5.1	Constructor	13
<b>4.6</b>	<b>Public Method Definitions</b>	<b>14</b>
4.6.1	Function deployPadawan	14
4.6.2	Function deployReserveProposal	14
4.6.3	Function getStats	15
4.6.4	Function getStored	15
4.6.5	Function getTotalDistributedCb	16
4.6.6	Function updateAddr	16
4.6.7	Function updateCode	16
<b>4.7</b>	<b>Internal Method Definitions</b>	<b>17</b>
4.7.1	Function _allCheckPassed	17
4.7.2	Function _beforeProposalDeploy	17
4.7.3	Function _createChecks	17
4.7.4	Function _deployProposals	18
4.7.5	Function _passCheck	18

---

In file `Demiurge.sol`

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

## 4.1 Contract Inheritance

Base	
PadawanResolver	
ProposalResolver	
IDemiurgeStoreCb	
IFaucetCb	

## 4.2 Constant Definitions

- OK

```

30  uint8 constant CHECK_PROPOSAL = 1;
31  uint8 constant CHECK_PADAWAN = 2;
33  uint128 constant TOTAL_EMISSION = 21000000;
```

## 4.3 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;
```

## 4.4 Modifier Definitions

### 4.4.1 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         tvn.accept();
69         -;
70     }

```

### 4.4.2 Modifier onlyStore

- OK

```

72     modifier onlyStore() {
73         require(msg.sender == _addrStore);
74         tvn.accept();
75         -;
76     }

```

## 4.5 Constructor Definitions

### 4.5.1 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() == tvn.pubkey(), 101);
85         }
86         require(addrStore != address(0), Errors.
87             STORE_SHOULD_BE_NOT_NULL);
88         tvn.accept();
89
90         if (addrStore != address(0)) {
91             _addrStore = addrStore;
92             DemiurgeStore(_addrStore).queryCode{value: 0.2 ton,
93                 bounce: true}(ContractType.Proposal);
94             DemiurgeStore(_addrStore).queryCode{value: 0.2 ton,
95                 bounce: true}(ContractType.Padawan);
96             DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
97                 bounce: true}(ContractAddr.DensRoot);
98             DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
99                 bounce: true}(ContractAddr.TokenRoot);
100             DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
101                 bounce: true}(ContractAddr.Faucet);
102         }
103         _createChecks();
104     }

```

## 4.6 Public Method Definitions

### 4.6.1 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
108             }(_addrTokenRoot);
109     }

```

### 4.6.2 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     }

```

### 4.6.3 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     }

```

### 4.6.4 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     }

```



### 4.6.5 Function getTotalDistributedCb

- | Critical issue:  | No permission check in<br><code>Demiurge.getTotalDistributedCb</code> |
|--|---|
| <ul style="list-style-type: none"> <li>Anybody can send this message. An attacker could use it to force the deployment of all proposals with a wrong number of total votes.</li> </ul> |   |
- 
- | Critical issue: No value check in <code>Demiurge.getTotalDistributedCb</code>  |
|--|
| <ul style="list-style-type: none"> <li>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.</li> </ul> |
- 
- 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     }

```

### 4.6.6 Function updateAddr

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

```

174     function updateAddr(ContractAddr kind, address addr) external
175         override onlyStore {
176         require(addr != address(0));
177         if (kind == ContractAddr.DensRoot) {
178             _addrDensRoot = addr;
179         } else if (kind == ContractAddr.TokenRoot) {
180             _addrTokenRoot = addr;
181         } else if (kind == ContractAddr.Faucet) {
182             _addrFaucet = addr;
183         }
184     }

```

### 4.6.7 Function updateCode

- OK

```

185     function updateCode(ContractType kind, TvmCell code) external
186         override onlyStore {
187         tvn.accept();
188         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 }

```

## 4.7 Internal Method Definitions

### 4.7.1 Function `_allCheckPassed`

- OK

```

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

```

### 4.7.2 Function `_beforeProposalDeploy`

- OK

```

134     function _beforeProposalDeploy(
135         uint8 i
136     ) private {
137         uint256 hashState = tvn.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,
142              _newProposals[i].specific);
143
144         IFaucet(_addrFaucet).getTotalDistributed
145             {value: 0.2 ton, flag: 1, bounce: false}();
146         _getBalancePendings += 1;
147     }

```

### 4.7.3 Function `_createChecks`

- OK

```

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

```

#### 4.7.4 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,
160                     value: START_BALANCE}(
161                     _totalVotes,
162                     _newProposals[i].addrClient,
163                     _newProposals[i].proposalType,
164                     _newProposals[i].specific,
165                     _newProposals[i].codePadawan
166                 );
167                 _deployedProposalsCounter++;
168             }
169             delete _newProposals;
170         }
```

#### 4.7.5 Function `_passCheck`

- OK

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

## Chapter 5

# Contract DemiurgeStore

### Contents

---

<b>5.1</b>	<b>Overview</b>	<b>19</b>
<b>5.2</b>	<b>General Minor-level Remarks</b>	<b>19</b>
<b>5.3</b>	<b>Public Functions</b>	<b>20</b>
5.3.1	Function queryAddr	20
5.3.2	Function queryCode	20
5.3.3	Function setDensRootAddr	20
5.3.4	Function setFaucetAddr	20
5.3.5	Function setPadawanCode	21
5.3.6	Function setProposalCode	21
5.3.7	Function setTokenRootAddr	21

---

### 5.1 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.

### 5.2 General Minor-level Remarks

In general, the infrastructure would be safer if this contract would be implemented in two phases:

- In the Initialization phase, the contract is waiting for all the `setXXX` methods to be called to initialize all the fields. A bitmap can be used to keep the current initialization state. Any attempt to user a `getXXX` method should fail.

- In the Post-Initialization phase, the contract accepts to reply to `getXXX` methods, but `setXXX` methods are disabled.

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

## 5.3 Public Functions

### 5.3.1 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,
46             bounce: false}(kind, addr);

```

### 5.3.2 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,
41             bounce: false}(kind, code);

```

### 5.3.3 Function `setDensRootAddr`

- OK

```

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

```

### 5.3.4 Function `setFaucetAddr`

- OK

```

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

```

### 5.3.5 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     }
```

### 5.3.6 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     }
```

### 5.3.7 Function setTokenRootAddr

- OK

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

## Chapter 6

# Contract Padawan

### Contents

---

<b>6.1</b>	<b>Overview</b>	<b>23</b>
<b>6.2</b>	<b>Static Variable Definitions</b>	<b>23</b>
<b>6.3</b>	<b>Variable Definitions</b>	<b>23</b>
<b>6.4</b>	<b>Modifier Definitions</b>	<b>23</b>
6.4.1	Modifier onlyOwner	23
6.4.2	Modifier onlyTokenRoot	24
<b>6.5</b>	<b>Constructor Definitions</b>	<b>24</b>
6.5.1	Constructor	24
<b>6.6</b>	<b>Public Method Definitions</b>	<b>24</b>
6.6.1	Function confirmVote	24
6.6.2	Function depositTokens	25
6.6.3	Function getActiveProposals	25
6.6.4	Function getAddresses	25
6.6.5	Function getAll	25
6.6.6	Function getTipAccount	26
6.6.7	Function getVoteInfo	26
6.6.8	Function onGetBalance	26
6.6.9	Function onTokenWalletDeploy	26
6.6.10	Function reclaimDeposit	26
6.6.11	Function rejectVote	27
6.6.12	Function updateStatus	27
6.6.13	Function vote	28
<b>6.7</b>	<b>Internal Method Definitions</b>	<b>29</b>
6.7.1	Function _createTokenAccount	29
6.7.2	Function _unlockDeposit	29
6.7.3	Function _updateLockedVotes	29

---

## 6.1 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.

## 6.2 Static Variable Definitions

- OK

```
18 address static _deployer;
19 address static _owner;
```

## 6.3 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;
```

## 6.4 Modifier Definitions

### 6.4.1 Modifier `onlyOwner`

- OK

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



### 6.4.2 Modifier onlyTokenRoot

- OK

```

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

```

## 6.5 Constructor Definitions

### 6.5.1 Constructor

- OK

```

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

```

## 6.6 Public Method Definitions

### 6.6.1 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     }

```

### 6.6.2 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     }

```

### 6.6.3 Function getActiveProposals

- OK

```

228     function getActiveProposals() public view returns (mapping(
229         address => uint32) activeProposals) {
230         activeProposals = _activeProposals;
231     }

```

### 6.6.4 Function getAddresses

- OK

```

224     function getAddresses() public view returns (address
225         ownerAddress) {
226         ownerAddress = _owner;
227     }

```

### 6.6.5 Function getAll

- OK

```

207     function getAll() external view returns (TipAccount tipAccount,
208         uint32 reqVotes, uint32 totalVotes, uint32 lockedVotes) {
209         tipAccount = _tipAccount;
210         reqVotes = _requestedVotes;
211         totalVotes = _totalVotes;
212         lockedVotes = _lockedVotes;
213     }

```

### 6.6.6 Function getTipAccount

- OK

```

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

```

### 6.6.7 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     }

```

### 6.6.8 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     }

```

### 6.6.9 Function onTokenWalletDeploy

- OK

```

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

```

### 6.6.10 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)
104         external onlyOwner {
105             require(msg.value >= 3 ton, Errors.MSG_VALUE_TOO_LOW);
106             require(votes <= _totalVotes, Errors.NOT_ENOUGH_VOTES);
107             require(returnTo != address(0));
108             _returnTo = returnTo;
109             _requestedVotes = votes;
110
111             if (_requestedVotes <= _totalVotes - _lockedVotes) {
112                 _unlockDeposit();
113             } else {
114                 _requestedVotes = 0;
115             }
116
117             optional(address, uint32) optActiveProposal =
118                 _activeProposals.min();
119             while (optActiveProposal.hasValue()) {
120                 (address addrActiveProposal,) = optActiveProposal.get();
121                 IProposal(addrActiveProposal).queryStatus
122                     {value: QUERY_STATUS_FEE, bounce: true, flag: 1}
123                     ();
124                 optActiveProposal = _activeProposals.next(
125                     addrActiveProposal);
126             }
127         }

```

### 6.6.11 Function rejectVote

- OK

```

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

```

### 6.6.12 Function updateStatus

- OK

```

127     function updateStatus(ProposalState state) external
128         onlyContract {
129             optional(uint32) optActiveProposal = _activeProposals.fetch
130                 (msg.sender);
131             require(optActiveProposal.hasValue());
132             tvmm.accept();
133
134             if (state >= ProposalState.Ended) {
135                 delete _activeProposals[msg.sender];
136                 _updateLockedVotes();
137             }
138
139             if (_requestedVotes != 0 && _requestedVotes <= _totalVotes
140                 - _lockedVotes) {
141                 _unlockDeposit();
142             }
143         }
144     }

```

### 6.6.13 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)
56         external onlyOwner {
57             require(msg.value >= VOTE_FEE, Errors.MSG_VALUE_TOO_LOW);
58             optional(uint32) optActiveProposal = _activeProposals.fetch
59                 (proposal);
60
61             uint32 activeProposalVotes = optActiveProposal.hasValue() ?
62                 optActiveProposal.get() : 0;
63             uint32 availableVotes = _totalVotes - activeProposalVotes;
64             require(votes <= availableVotes, Errors.NOT_ENOUGH_VOTES);
65
66             // TODO: better to remove
67             if (activeProposalVotes == 0) {
68                 _activeProposals[proposal] = 0;
69             }
70
71             IProposal(proposal).vote
72                 {value: 0, flag: 64, bounce: true}

```

```

70         (_owner, choice, votes);
71     }

```

## 6.7 Internal Method Definitions

### 6.7.1 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(
201             this).value, 1 ton);

```

### 6.7.2 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     }

```

### 6.7.3 Function `_updateLockedVotes`

- OK

```

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

```

## Chapter 7

# Contract PadawanResolver

### Contents

<b>7.1 Overview</b>	<b>30</b>
<b>7.2 Variable Definitions</b>	<b>30</b>
<b>7.3 Public Method Definitions</b>	<b>30</b>
7.3.1 Function resolvePadawan	30
<b>7.4 Internal Method Definitions</b>	<b>31</b>
7.4.1 Function _buildPadawanState	31

## 7.1 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.

## 7.2 Variable Definitions

- OK

```
8 TvmCell _codePadawan;
```

## 7.3 Public Method Definitions

### 7.3.1 Function resolvePadawan

- OK

```

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

```

## 7.4 Internal Method Definitions

### 7.4.1 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
17         view returns (TvmCell) {
18         return tvm.buildStateInit({
19             contr: Padawan,
20             varInit: {_deployer: address(this), _owner: owner},
21             code: _codePadawan
22         });
23     }

```



## Chapter 8

# Contract Proposal

### Contents

---

<b>8.1</b>	<b>Overview</b>	<b>32</b>
<b>8.2</b>	<b>Static Variable Definitions</b>	<b>33</b>
<b>8.3</b>	<b>Variable Definitions</b>	<b>33</b>
<b>8.4</b>	<b>Constructor Definitions</b>	<b>33</b>
8.4.1	Constructor	33
<b>8.5</b>	<b>Public Method Definitions</b>	<b>34</b>
8.5.1	Function <code>getAll</code>	34
8.5.2	Function <code>getCurrentVotes</code>	34
8.5.3	Function <code>getInfo</code>	34
8.5.4	Function <code>getVotingResults</code>	34
8.5.5	Function <code>queryStatus</code>	35
8.5.6	Function <code>vote</code>	35
8.5.7	Function <code>wrapUp</code>	36
<b>8.6</b>	<b>Internal Method Definitions</b>	<b>36</b>
8.6.1	Function <code>_buildPadawanState</code>	36
8.6.2	Function <code>_calculateVotes</code>	36
8.6.3	Function <code>_changeState</code>	36
8.6.4	Function <code>_finalize</code>	37
8.6.5	Function <code>_softMajority</code>	37
8.6.6	Function <code>_tryEarlyComplete</code>	38
8.6.7	Function <code>_wrapUp</code>	38

---

## 8.1 Overview

In file `Proposal.sol`

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

## 8.2 Static Variable Definitions

- OK

```
13 address static _deployer;
14 string static _title;
```

## 8.3 Variable Definitions

- OK

```
16 address public _addrClient;
18 ProposalInfo public _proposalInfo;
20 ProposalResults _results;
21 VoteCountModel _voteCountModel;
```

## 8.4 Constructor Definitions

### 8.4.1 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 }

```

## 8.5 Public Method Definitions

### 8.5.1 Function getAll

- OK

```

168     function getAll() public view override returns (ProposalInfo
169         info) {
170         info = _proposalInfo;
171     }

```

### 8.5.2 Function getCurrentVotes

- OK

```

181     function getCurrentVotes() external override view returns (
182         uint32 votesFor, uint32 votesAgainst) {
183         return (_proposalInfo.votesFor, _proposalInfo.votesAgainst)
184         ;
185     }

```

### 8.5.3 Function getInfo

- OK

```

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

```

### 8.5.4 Function getVotingResults

- OK

```

172     function getVotingResults() public view returns (
173         ProposalResults vr) {
174         require(_proposalInfo.state > ProposalState.Ended, Errors.
175             VOTING_HAS_NOT_ENDED);
176         vr = _results;
177     }

```

### 8.5.5 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     }

```

### 8.5.6 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`, together with `rejectVote` or `confirmVote` messages, because of the flag 64. Need to test that it is ok.

```

55     function vote(address addrPadawanOwner, bool choice, uint32
56         votesCount) external override {
57         address addrPadawan = resolvePadawan(addrPadawanOwner);
58         uint16 errorCode = 0;
59
60         if (addrPadawan != msg.sender) {
61             errorCode = Errors.NOT_AUTHORIZED_CONTRACT;
62         } else if (now < _proposalInfo.start) {
63             errorCode = Errors.VOTING_NOT_STARTED;
64         } else if (now > _proposalInfo.end) {
65             errorCode = Errors.VOTING_HAS_ENDED;
66         }
67
68         if (errorCode > 0) {
69             IPadawan(msg.sender).rejectVote{value: 0, flag: 64,
70                 bounce: true}(votesCount, errorCode);
71         } else {
72             IPadawan(msg.sender).confirmVote{value: 0, flag: 64,
73                 bounce: true}(votesCount);
74             if (choice) {
75                 _proposalInfo.votesFor += votesCount;
76             } else {
77                 _proposalInfo.votesAgainst += votesCount;
78             }
79         }
80     }
81     _wrapUp();
82 }

```

### 8.5.7 Function wrapUp

- OK

```

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

```

## 8.6 Internal Method Definitions

### 8.6.1 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
155         override returns (TvmCell) {
156         return tvm.buildStateInit({
157             contr: Padawan,
158             varInit: {_deployer: _deployer, _owner: owner},
159             code: _codePadawan
160         });
161     }

```

### 8.6.2 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     }

```

### 8.6.3 Function \_changeState

- OK

```

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

```

### 8.6.4 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 :
93         ProposalState.NotPassed;
94
95     _changeState(state);
96
97     IClient(address(_addrClient)).onProposalPassed{value: 1 ton
98         } (_proposalInfo);
99
100    emit ProposalFinalized(_results);
101 }

```

### 8.6.5 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 *
147         (( _proposalInfo.totalVotes / 2) - ( _proposalInfo.
148             totalVotes / 10))) / ( _proposalInfo.totalVotes / 2);
149     return passed;
150 }

```

### 8.6.6 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 }

```

### 8.6.7 Function `_wrapUp`

- Minor issue: the function could immediately check if the state is above `Ended` to avoid recomputing again;
- 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,
122                                 _proposalInfo.votesAgainst);
123     } else {
124         (completed, passed) = _tryEarlyComplete(_proposalInfo.
125                                                 votesFor, _proposalInfo.votesAgainst);
126     }
127
128     if (completed) {
129         _changeState(ProposalState.Ended);
130         _finalize(passed);
131     }
132 }

```

## Chapter 9

# Contract ProposalResolver

### Contents

<b>9.1 Overview</b>	<b>39</b>
<b>9.2 Variable Definitions</b>	<b>39</b>
<b>9.3 Public Method Definitions</b>	<b>39</b>
9.3.1 Function resolveProposal	39
<b>9.4 Internal Method Definitions</b>	<b>40</b>
9.4.1 Function _buildProposalState	40

### 9.1 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.

### 9.2 Variable Definitions

- OK

```
6 TvmCell _codeProposal;
```

### 9.3 Public Method Definitions

#### 9.3.1 Function resolveProposal

- OK



```

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

```

## 9.4 Internal Method Definitions

### 9.4.1 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, only `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 tvn.buildStateInit({
16             contr: Proposal,
17             varInit: {_deployer: address(this), _title: title},
18             code: _codeProposal
19         });
20     }

```