Audit of the DENS-SMV Project

By OCamlPro

August 20, 2021

Table of Major and Critical Issues

Contents

Chapter 1

Introduction

1.0.1 Location

The Location section should be read as: The source code is available at `https://github.com/RSquad/dens-smv` at branch master with hash code equal to fbdfe4bca3c372b02cacf9788b4ad37112d0da2c and `https://github.com/RSquad/BFTG` (SMV part only) at branch master with hash code equal to 7c6ec7d811bcc1f228a3499ab19f6d20652ca94b

1.0.2 End Date

The contest ends at Aug 20, 2021, 23:59:59 UTC

Chapter 2

Overview

Chapter 3

Contract Base

Contents

In file `Base.sol`

3.1 Constant Definitions

```
    uint16 constant ERROR_DIFFERENT_CALLER =  211;

    uint64 constant START_BALANCE       = 3 ton;

    uint64 constant DEPLOYER_FEE        = 0.1 ton;

    uint64 constant PROCESS_FEE         = 0.3 ton;

    uint64 constant VOTE_FEE            = 1 ton;

    uint64 constant DEPLOY_FEE          = START_BALANCE +
        DEPLOYER_FEE;

    uint64 constant DEPLOY_PAY          = DEPLOY_FEE + PROCESS_FEE;

    uint64 constant DEPLOY_PROPOSAL_FEE = 5 ton;

    uint64 constant DEPLOY_PROPOSAL_PAY = DEPLOY_PROPOSAL_FEE +
        PROCESS_FEE;
    uint64 constant DEPOSIT_TONS_FEE    = 1 ton;

    uint64 constant DEPOSIT_TONS_PAY    = DEPOSIT_TONS_FEE + PROCESS_FEE;

    uint64 constant DEPOSIT_TOKENS_FEE  = 0.5 ton + DEPOSIT_TONS_FEE;

    uint64 constant DEPOSIT_TOKENS_PAY  = DEPOSIT_TOKENS_FEE + PROCESS_FEE;

    uint64 constant TOKEN_ACCOUNT_FEE   = 2 ton;

    uint64 constant TOKEN_ACCOUNT_PAY   = TOKEN_ACCOUNT_FEE + PROCESS_FEE;

    uint64 constant QUERY_STATUS_FEE    = 0.02 ton;

    uint64 constant QUERY_STATUS_PAY    = QUERY_STATUS_FEE + DEF_RESPONSE_VALUE;

    uint64 constant DEF_RESPONSE_VALUE = 0.03 ton;

    uint64 constant DEF_COMPUTE_VALUE = 0.2 ton;
```

3.2 Modifier Definitions

3.2.1 Modifier signed

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

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.

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

3.2.3 Modifier onlyContract

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

3.2.4 Modifier onlyMe

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

Chapter 4

Contract Demiurge

Contents

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

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

4.3 Variable Definitions

- OK

```
    uint32 _deployedPadawansCounter = 0;
    uint32 _deployedProposalsCounter = 0;
    uint16 _version = 3;
    address _addrStore;
    address _addrDensRoot;
    address _addrTokenRoot;
    address _addrFaucet;
    uint8 _checkList;
    NewProposal[] public _newProposals;
    uint8 public _getBalancePendings = 0;
    uint128 public _totalVotes = 0;
```

4.4 Modifier Definitions

4.4.1 Modifier checksEmpty

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

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

4.4.2 Modifier onlyStore

- OK

```
    modifier onlyStore() {
        require(msg.sender == _addrStore);
        tvm.accept();
        _;
    }
```

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.

```
    constructor(address addrStore) public {
        if (msg.sender == address(0)) {
            require(msg.pubkey() == tvm.pubkey(), 101);
        }
        require(addrStore != address(0), Errors.
            STORE_SHOULD_BE_NOT_NULL);
        tvm.accept();

        if (addrStore != address(0)) {
            _addrStore = addrStore;
            DemiurgeStore(_addrStore).queryCode{value: 0.2 ton,
                bounce: true}(ContractType.Proposal);
            DemiurgeStore(_addrStore).queryCode{value: 0.2 ton,
                bounce: true}(ContractType.Padawan);
            DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
                bounce: true}(ContractAddr.DensRoot);
            DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
                bounce: true}(ContractAddr.TokenRoot);
            DemiurgeStore(_addrStore).queryAddr{value: 0.2 ton,
                bounce: true}(ContractAddr.Faucet);
        }

        _createChecks();
    }
```

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.

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

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.

```
    function deployReserveProposal(
        string title,
        ReserveProposalSpecific specific
    ) external onlyContract {
        require(msg.value >= DEPLOY_PROPOSAL_FEE);
        TvmBuilder b;
        b.store(specific);
        TvmCell cellSpecific = b.toCell();

        NewProposal _newProposal = NewProposal(
            0,
            _addrDensRoot,
            ProposalType.Reserve,
            cellSpecific,
            _codePadawan,
            _buildProposalState(title)
        );
        _newProposals.push(_newProposal);

        _beforeProposalDeploy(uint8(_newProposals.length - 1));
    }
```

4.6.3 Function getStats

- OK

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

4.6.4 Function getStored

- OK

```
    function getStored() public view returns (
        TvmCell codePadawan,
        TvmCell codeProposal,
        address addrStore,
        address addrDensRoot,
        address addrTokenRoot,
        address addrFaucet
    ) {
        codePadawan = _codePadawan;
        codeProposal = _codeProposal;
        addrStore = _addrStore;
        addrDensRoot = _addrDensRoot;
        addrTokenRoot = _addrTokenRoot;
        addrFaucet = _addrFaucet;
    }
```

4.6.5 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.

```
    function getTotalDistributedCb(
        uint128 totalDistributed
    ) public override {
        _totalVotes = totalDistributed;
        _getBalancePendings -= 1;
        _deployProposals();
    }
```

4.6.6 Function updateAddr

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

```
    function updateAddr(ContractAddr kind, address addr) external
        override onlyStore {
        require(addr != address(0));
        if (kind == ContractAddr.DensRoot) {
            _addrDensRoot = addr;
        } else if (kind == ContractAddr.TokenRoot) {
            _addrTokenRoot = addr;
        } else if (kind == ContractAddr.Faucet) {
            _addrFaucet = addr;
        }
    }
```

4.6.7 Function updateCode

- OK

```
    function updateCode(ContractType kind, TvmCell code) external
        override onlyStore {
        tvm.accept();
        if (kind == ContractType.Proposal) {
            _codeProposal = code;
            _passCheck(CHECK_PROPOSAL);
        } else if (kind == ContractType.Padawan) {
            _codePadawan = code;
            _passCheck(CHECK_PADAWAN);
        }
    }
```

4.7 Internal Method Definitions

4.7.1 Function _allCheckPassed

- OK

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

4.7.2 Function _beforeProposalDeploy

- OK

```
    function _beforeProposalDeploy(
        uint8 i
    ) private {
        uint256 hashState = tvm.hash(_newProposals[i].state);
        address addrProposal = address.makeAddrStd(0, hashState);
        IClient(_addrDensRoot).onProposalDeploy
            {value: 1 ton, bounce: true}
            (addrProposal, _newProposals[i].proposalType,
                _newProposals[i].specific);

        IFaucet(_addrFaucet).getTotalDistributed
            {value: 0.2 ton, flag: 1, bounce: false}();
        _getBalancePendings += 1;
    }
```

4.7.3 Function _createChecks

- OK

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

4.7.4 Function _deployProposals

- OK

```
    function _deployProposals() private {
        if(_getBalancePendings == 0) {
            for(uint8 i = 0; i < _newProposals.length; i++) {
                new Proposal {stateInit: _newProposals[i].state,
                    value: START_BALANCE}(
                    _totalVotes,
                    _newProposals[i].addrClient,
                    _newProposals[i].proposalType,
                    _newProposals[i].specific,
                    _newProposals[i].codePadawan
                );
                _deployedProposalsCounter++;
            }
            delete _newProposals;
        }
    }
```

4.7.5 Function _passCheck

- OK

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

Chapter 5

Contract DemiurgeStore

Contents

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-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).

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.

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

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

5.3.3 Function setDensRootAddr

- OK

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

5.3.4 Function setFaucetAddr

- OK

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

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`

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

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`

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

5.3.7 Function setTokenRootAddr

- OK

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

Chapter 6

Contract Padawan

Contents

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

```
    address static _deployer;
```

```
    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.

```
    address _addrTokenRoot;
```

```
    TipAccount _tipAccount;
```

```
    address _returnTo;
```

```
    mapping(address => uint32) _activeProposals;
```

```
    uint32 _requestedVotes;
```

```
    uint32 _totalVotes;
```

```
    uint32 _lockedVotes;
```

6.4 Modifier Definitions

6.4.1 Modifier onlyOwner

- OK

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

6.4.2 Modifier onlyTokenRoot

- OK

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

6.5 Constructor Definitions

6.5.1 Constructor

- OK

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

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.

```
    function confirmVote(uint32 votesCount) external onlyContract {
        // TODO: better to check is it proposal or not
        optional(uint32) optActiveProposal = _activeProposals.fetch
            (msg.sender);
        require(optActiveProposal.hasValue());

        _activeProposals[msg.sender] += votesCount;

        _updateLockedVotes();

        _owner.transfer(0, false, 64);
    }
```

6.6.2 Function depositTokens

- OK

```
    function depositTokens() external onlyOwner view {
        require(msg.value >= DEPOSIT_TOKENS_FEE, Errors.
            MSG_VALUE_TOO_LOW);
        require(_tipAccount.addr != address(0), Errors.
            ACCOUNT_DOES_NOT_EXIST);

        ITokenWallet(_tipAccount.addr).getBalance_InternalOwner
            {value: 0, flag: 64, bounce: true}
            (tvm.functionId(onGetBalance));
    }
```

6.6.3 Function getActiveProposals

- OK

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

6.6.4 Function getAddresses

- OK

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

6.6.5 Function getAll

- OK

```
    function getAll() external view returns (TipAccount tipAccount,
         uint32 reqVotes, uint32 totalVotes, uint32 lockedVotes) {
        tipAccount = _tipAccount;
        reqVotes = _requestedVotes;
        totalVotes =  _totalVotes;
        lockedVotes = _lockedVotes;
    }
```

6.6.6 Function getTipAccount

- OK

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

6.6.7 Function getVoteInfo

- OK

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

6.6.8 Function onGetBalance

- OK

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

6.6.9 Function onTokenWalletDeploy

- OK

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

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

```
    function reclaimDeposit(uint32 votes, address returnTo) external onlyOwner {
        require(msg.value >= 3 ton, Errors.MSG_VALUE_TOO_LOW);
        require(votes <= _totalVotes, Errors.NOT_ENOUGH_VOTES);
        require(returnTo != address(0));
        _returnTo = returnTo;
        _requestedVotes = votes;

        if (_requestedVotes <= _totalVotes - _lockedVotes) {
            _unlockDeposit();
        } else {
            _requestedVotes = 0;
        }

        optional(address, uint32) optActiveProposal = _activeProposals.min();
        while (optActiveProposal.hasValue()) {
            (address addrActiveProposal,) = optActiveProposal.get();
            IProposal(addrActiveProposal).queryStatus
                {value: QUERY_STATUS_FEE, bounce: true, flag: 1}
                ();
            optActiveProposal = _activeProposals.next(addrActiveProposal);
        }
    }
```

6.6.11 Function rejectVote

- OK

```
    function rejectVote(uint32 votesCount, uint16 errorCode) external onlyContract {
        votesCount; errorCode;

        // TODO: better to check is it proposal or not
        optional(uint32) optActiveProposal = _activeProposals.fetch(msg.sender);
        require(optActiveProposal.hasValue());
        uint32 activeProposalVotes = optActiveProposal.get();
        if (activeProposalVotes == 0) {
            delete _activeProposals[msg.sender];
        }

        _owner.transfer(0, false, 64);
    }
```

6.6.12 Function updateStatus

- OK

```
    function updateStatus(ProposalState state) external
        onlyContract {
        optional(uint32) optActiveProposal = _activeProposals.fetch
            (msg.sender);
        require(optActiveProposal.hasValue());
        tvm.accept();

        if (state >= ProposalState.Ended) {
            delete _activeProposals[msg.sender];
            _updateLockedVotes();
        }

        if (_requestedVotes != 0 && _requestedVotes <= _totalVotes
            - _lockedVotes) {
            _unlockDeposit();
        }
    }
```

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.

```
    function vote(address proposal, bool choice, uint32 votes)
        external onlyOwner {
        require(msg.value >= VOTE_FEE, Errors.MSG_VALUE_TOO_LOW);
        optional(uint32) optActiveProposal = _activeProposals.fetch
            (proposal);

        uint32 activeProposalVotes = optActiveProposal.hasValue() ?
             optActiveProposal.get() : 0;
        uint32 availableVotes = _totalVotes - activeProposalVotes;
        require(votes <= availableVotes, Errors.NOT_ENOUGH_VOTES);

        // TODO: better to remove
        if (activeProposalVotes == 0) {
            _activeProposals[proposal] = 0;
        }

        IProposal(proposal).vote
            {value: 0, flag: 64, bounce: true}
            (_owner, choice, votes);
    }
```

6.7 Internal Method Definitions

6.7.1 Function _createTokenAccount

- OK

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

6.7.2 Function _unlockDeposit

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

```
    function _unlockDeposit() private {
        ITokenWallet(_tipAccount.addr).transfer
            {value: 0.1 ton + 0.1 ton}
            (_returnTo, _requestedVotes, 0.1 ton);
        _totalVotes -= _requestedVotes;
        _requestedVotes = 0;
        _returnTo = address(0);
    }
```

6.7.3 Function _updateLockedVotes

- OK

```
    function _updateLockedVotes() private inline {
        optional(address, uint32) optActiveProposal =
            _activeProposals.min();
        uint32 lockedVotes;
        while (optActiveProposal.hasValue()) {
            (address addr, uint32 votes) = optActiveProposal.get();
            if (votes > lockedVotes) {
                lockedVotes = votes;
            }
            optActiveProposal = _activeProposals.next(addr);
        }
        _lockedVotes = lockedVotes;
    }
```

Chapter 7

Contract PadawanResolver

Contents

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

```
    TvmCell _codePadawan;
```

7.3 Public Method Definitions

7.3.1 Function resolvePadawan

- OK

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

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.

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

Chapter 8

Contract Proposal

Contents

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

```
    address static _deployer;
```

```
    string static _title;
```

8.3 Variable Definitions

- OK

```
    address public _addrClient;
```

```
    ProposalInfo public _proposalInfo;
```

```
    ProposalResults _results;
```

```
    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.

```
    constructor(
        uint128 totalVotes,
        address addrClient,
        ProposalType proposalType,
        TvmCell specific,
        TvmCell codePadawan
    ) public {
        require(_deployer == msg.sender);

        _addrClient = addrClient;

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

        _codePadawan = codePadawan;

        _voteCountModel = VoteCountModel.SoftMajority;
    }
```

8.5 Public Method Definitions

8.5.1 Function getAll

- OK

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

8.5.2 Function getCurrentVotes

- OK

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

8.5.3 Function getInfo

- OK

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

8.5.4 Function getVotingResults

- OK

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

8.5.5 Function queryStatus

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

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

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` (in `_wrapUp`), together with `rejectVote` or `confirmVote` messages, because of the flag 64. Need to test what happens if two messages are sent by the same transaction, with one of them containing the flag 64.

```
    function vote(address addrPadawanOwner, bool choice, uint32 votesCount) external override {
        address addrPadawan = resolvePadawan(addrPadawanOwner);
        uint16 errorCode = 0;

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

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

        _wrapUp();
    }
```

8.5.7 Function wrapUp

- OK

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

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.

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

8.6.2 Function _calculateVotes

- OK

```
    function _calculateVotes(
        uint32 yes,
        uint32 no
    ) private view returns (bool) {
        bool passed = false;
        passed = _softMajority(yes, no);
        return passed;
    }
```

8.6.3 Function _changeState

- OK

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

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`

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

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

        _changeState(state);

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

        emit ProposalFinalized(_results);
    }
```

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.

```
    function _softMajority(
        uint32 yes,
        uint32 no
    ) private view returns (bool) {
        bool passed = false;
        passed = yes >= 1 + (_proposalInfo.totalVotes / 10) + (no *
             ((_proposalInfo.totalVotes / 2) - (_proposalInfo.
            totalVotes / 10))) / (_proposalInfo.totalVotes / 2);
        return passed;
    }
```

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.

```
    function _tryEarlyComplete(
        uint32 yes,
        uint32 no
    ) private view returns (bool, bool) {
        (bool completed, bool passed) = (false, false);
        if (yes * 2 > _proposalInfo.totalVotes) {
            completed = true;
            passed = true;
        } else if(no * 2 >= _proposalInfo.totalVotes) {
            completed = true;
            passed = false;
        }
        return (completed, passed);
    }
```

8.6.7 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;

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

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

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

Chapter 9

Contract ProposalResolver

Contents

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

```
  TvmCell _codeProposal;
```

9.3 Public Method Definitions

9.3.1 Function resolveProposal

- OK

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

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, 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.

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