Audit

By OCamlPro

September 9, 2021

Table of Major and Critical Issues

Contents

Chapter 1

Only for Auditors

1.1 To edit this documents

In the report.tex file, choose:

- **\soldraftfalse** to remove draft mode (watermarks, advises)
- **\solmodulestrue** to display modules by chapter instead of contracts
- **\soltablestrue** 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}**

1.2 General Auditing Rules

- Check that types have the correct integer types (Pubkey : uint256, Amount: uint128, Time: uint64).
- Naming conventions: constants should for example be all uppercase, static variables should start with a prefix like `s_`, globals should start with a prefix like `g_` or `m_`, internal functions should start with a prefix `_`.
- Numbers should not appear in source, but be defined as constants.
- In constant definitions, verify that 2 consecutive errors have not the same error (common copy-paste error)

- Constants for amounts should be expressed in `ton` to prevent too many zeroes.
- Modifiers with `tvm.accept` must always check the source of the message
- Constructors with arguments must always check the source of the message to prevent anybody from calling the constructor and set variables instead of the real owner
- Failures should never happen after `tvm.accept` (such as `require`, division by zero, overflows, etc.)
- Most arguments should be protected by a `require`
- Before sending a message, the function should check that it has enough gas (to prevent a partial failure during the message sending phase)
- `tvm.accept` should only be called after verifying that the sender of the message if the contracts' owner

Chapter 2

Introduction

Chapter 3

Overview

Chapter 4

Contract Data

Contents

4.1 Module "IData.sol" 11
4.1.1 Pragmas 11
4.1.2 Contract Definitions 11
4.2 Module "IIndex.sol" 12
4.2.1 Pragmas 12
4.2.2 Contract Definitions 12
4.3 Module "IIndexBasis.sol" 13
4.3.1 Pragmas 13
4.3.2 Contract Definitions 13

4.1 Overview

In file `Data.sol`

4.2 Contract Inheritance

IData	
IndexResolver	

4.3 Static Variable Definitions

uint256	_id	

```
    uint256 static _id;
```

4.4 Variable Definitions

address	_addrRoot	
		used in @7.Data.transferOwnership
		used in @7.Data.getInfo
		used in @7.Data.deployIndex
		used in @7.Data.deployIndex
		used in @7.Data.deployIndex
		assigned in @7.Data.:constructor
		used in @7.Data.:constructor
address	_addrOwner	
		assigned in @7.Data.transferOwnership
		used in @7.Data.transferOwnership
		used in @7.Data.transferOwnership
		used in @7.Data.transferOwnership
		used in @7.Data.transferOwnership
		used in @7.Data.getOwner
		used in @7.Data.getInfo
		assigned in @7.Data.:constructor
		used in @7.Data.:constructor
address	_addrAuthor	
		assigned in @7.Data.:constructor
		used in @7.Data.:constructor

```
    address _addrRoot;
```

```
    address _addrOwner;
```

```
    address _addrAuthor;
```

4.5 Constructor Definitions

4.5.1 Constructor

Minor issue: Constants

Value ”101” should be defined as a constant

Major issue: addrOwner may be null

The owner of the contract may be null.

```
    constructor(address addrOwner, TvmCell codeIndex) public {
        optional(TvmCell) optSalt = tvm.codeSalt(tvm.code());
        require(optSalt.hasValue(), 101);
        (address addrRoot) = optSalt.get().toSlice().decode(address
            );
        require(msg.sender == addrRoot);
        require(msg.value >= Constants.MIN_FOR_DEPLOY);
        tvm.accept();
        _addrRoot = addrRoot;
        _addrOwner = addrOwner;
        _addrAuthor = addrOwner;
        _codeIndex = codeIndex;

        deployIndex(addrOwner);
    }
```

4.6 Public Method Definitions

4.6.1 Function getInfo

```
    function getInfo() public view override returns (
        address addrRoot,
        address addrOwner,
        address addrData
    ) {
        addrRoot = _addrRoot;
        addrOwner = _addrOwner;
        addrData = address(this);
    }
```

4.6.2 Function getOwner

```
    function getOwner() public view override returns(address
        addrOwner) {
        addrOwner = _addrOwner;
    }
```

4.6.3 Function transferOwnership

Critical issue: Methods called without value TODO: CHECK

IIndex(_).destruct() are called without sending funds.

Major issue: New owner may be null

The new owner of the contract may be null.

Minor issue: New owner may be equal to the old one

The new owner of the contract may be equal to the old one, hence destructing and rebuilding identical contracts.

```
    function transferOwnership(address addrTo) public override {
        require(msg.sender == _addrOwner);
        require(msg.value >= Constants.MIN_FOR_DEPLOY);

        address oldIndexOwner = resolveIndex(_addrRoot, address(
            this), _addrOwner);
        IIndex(oldIndexOwner).destruct();
        address oldIndexOwnerRoot = resolveIndex(address(0),
            address(this), _addrOwner);
        IIndex(oldIndexOwnerRoot).destruct();

        _addrOwner = addrTo;

        deployIndex(addrTo);
    }
```

4.7 Internal Method Definitions

4.7.1 Function deployIndex

```
    function deployIndex(address owner) private {
        TvmCell codeIndexOwner = _buildIndexCode(_addrRoot, owner);
        TvmCell stateIndexOwner = _buildIndexState(codeIndexOwner,
            address(this));
        new Index{stateInit: stateIndexOwner, value: 0.4 ton}(
            _addrRoot);

        TvmCell codeIndexOwnerRoot = _buildIndexCode(address(0),
            owner);
        TvmCell stateIndexOwnerRoot = _buildIndexState(
            codeIndexOwnerRoot, address(this));
        new Index{stateInit: stateIndexOwnerRoot, value: 0.4 ton}(
            _addrRoot);
    }
```

Chapter 5

Contract DataResolver

Contents

5.1 Module "Data.sol" 15
5.1.1 Pragmas 15
5.1.2 Imports 15
5.1.3 Contract Definitions 15
5.2 Module "DataResolver.sol" 16
5.2.1 Pragmas 16
5.2.2 Imports 16
5.2.3 Contract Definitions 16
5.3 Module "Index.sol" 17
5.3.1 Pragmas 17
5.3.2 Imports 17
5.3.3 Contract Definitions 17
5.4 Module "IndexBasis.sol" 18
5.4.1 Pragmas 18
5.4.2 Imports 18
5.4.3 Contract Definitions 18
5.5 Module "IndexResolver.sol" 19
5.5.1 Pragmas 19
5.5.2 Imports 19
5.5.3 Contract Definitions 19
5.6 Module "Manager.sol" 20
5.6.1 Pragmas 20
5.6.2 Imports 20
5.6.3 Contract Definitions 20
5.7 Module "NftRoot.sol" 21
5.7.1 Pragmas 21
5.7.2 Imports 21
5.7.3 Contract Definitions 21

5.1 Overview

In file `DataResolver.sol`

5.2 Variable Definitions

TvmCell	_codeData	
		assigned in @1.NftRoot.:constructor
		used in @1.NftRoot.:constructor
		used in @5.DataResolver._buildDataCode

```
    TvmCell _codeData;
```

5.3 Public Method Definitions

5.3.1 Function resolveCodeHashData

- TODO

```
    function resolveCodeHashData() public view returns (uint256
        codeHashData) {
        return tvm.hash(_buildDataCode(address(this)));
    }
```

5.3.2 Function resolveData

- TODO

```
    function resolveData(
        address addrRoot,
        uint256 id
    ) public view returns (address addrData) {
        TvmCell code = _buildDataCode(addrRoot);
        TvmCell state = _buildDataState(code, id);
        uint256 hashState = tvm.hash(state);
        addrData = address.makeAddrStd(0, hashState);
    }
```

5.4 Internal Method Definitions

5.4.1 Function _buildDataCode

- TODO

```
    function _buildDataCode(address addrRoot) internal virtual view
         returns (TvmCell) {
        TvmBuilder salt;
        salt.store(addrRoot);
        return tvm.setCodeSalt(_codeData, salt.toCell());
    }
```

5.4.2 Function _buildDataState

- TODO

```
    function _buildDataState(
        TvmCell code,
        uint256 id
    ) internal virtual pure returns (TvmCell) {
        return tvm.buildStateInit({
            contr: Data,
            varInit: {_id: id},
            code: code
        });
    }
```

Chapter 6

Contract Index

Contents

6.1 Overview 22
6.2 Contract Inheritance 22
6.3 Static Variable Definitions 22
6.4 Variable Definitions 23
6.5 Constructor Definitions 24
6.5.1 Constructor 24
6.6 Public Method Definitions 25
6.6.1 Function destruct 25
6.6.2 Function getInfo 25
6.6.3 Function getOwner 26
6.7 Internal Method Definitions 26
6.7.1 Function deployIndex 26

6.1 Overview

In file `Index.sol`

6.2 Contract Inheritance

IIndex	

6.3 Static Variable Definitions

address	_addrData	
		used in @8.Index.getInfo
		used in @8.Index.destruct
		used in @8.Index.destruct
		used in @8.Index.:constructor

```
    address static _addrData;
```

6.4 Variable Definitions

address	_addrRoot	
		used in @8.Index.getInfo
		assigned in @8.Index.:constructor
		used in @8.Index.:constructor
		assigned in @8.Index.:constructor
		used in @8.Index.:constructor
address	_addrOwner	
		used in @8.Index.getInfo
		assigned in @8.Index.:constructor
		used in @8.Index.:constructor

```
    address _addrRoot;
```

```
    address _addrOwner;
```

6.5 Constructor Definitions

6.5.1 Constructor

Minor issue: Constants
Values ”101” and ”address(0)” should be constants.

Minor issue: Double initialization of
_addrRoot is initialized twice if addrRoot = 0.

```
    constructor(address root) public {
        optional(TvmCell) optSalt = tvm.codeSalt(tvm.code());
        require(optSalt.hasValue(), 101);
        (address addrRoot, address addrOwner) = optSalt
            .get()
            .toSlice()
            .decode(address, address);
        require(msg.sender == _addrData);
        tvm.accept();
        _addrRoot = addrRoot;
        _addrOwner = addrOwner;
        if(addrRoot == address(0)) {
            _addrRoot = root;
        }
    }
```

6.6 Public Method Definitions

6.6.1 Function destruct

```
    function destruct() public override {
        require(msg.sender == _addrData);
        selfdestruct(_addrData);
    }
```

6.6.2 Function getInfo

```
    function getInfo() public view override returns (
        address addrRoot,
        address addrOwner,
        address addrData
    ) {
        addrRoot = _addrRoot;
        addrOwner = _addrOwner;
        addrData = _addrData;
    }
```

Chapter 7

Contract IndexBasis

Contents

7.1 Overview **27**
7.2 Variable Definitions **27**
7.3 Public Method Definitions **27**
7.3.1 Function resolveCodeHashData 27
7.3.2 Function resolveData 27
7.4 Internal Method Definitions **28**
7.4.1 Function _buildDataCode 28
7.4.2 Function _buildDataState 28

7.1 Overview

In file `IndexBasis.sol`

7.2 Static Variable Definitions

address	_addrRoot	
		used in @2.IndexBasis.getInfo
		used in @2.IndexBasis.destruct
uint256	_codeHashData	
		used in @2.IndexBasis.getInfo

```
    address static _addrRoot;
```

```
    uint256 static _codeHashData;
```

7.3 Modifier Definitions

7.3.1 Modifier onlyRoot

Minor issue: Modifiers

Modifiers are often source of bugs ; using them should be avoided (especially when containing calls to tvm.accept()).

Minor issue: Constants

Value ”100” should be defined as a constant.

```
    modifier onlyRoot() {
        require(msg.sender == _addrRoot, 100);
        tvm.accept();
        _;
    }
```

7.4 Constructor Definitions

7.4.1 Constructor

- TODO

```
    constructor() public onlyRoot {}
```

7.5 Public Method Definitions

7.5.1 Function destruct

Minor issue: Superfluous tvm.accept()

The function tvm.accept() do not need to be called (unless _addrRoot = 0, which should not be the case).

```
    function destruct() public onlyRoot {
        selfdestruct(_addrRoot);
    }
```

7.5.2 Function getInfo

```
    function getInfo() public view returns (address addrRoot,
        uint256 codeHashData) {
        addrRoot = _addrRoot;
        codeHashData = _codeHashData;
    }
```

Chapter 8

Contract IndexResolver

Contents

8.1 Overview 29
8.2 Contract Inheritance 29
8.3 Static Variable Definitions 29
8.4 Variable Definitions 30
8.5 Constructor Definitions 30
8.5.1 Constructor 30
8.6 Public Method Definitions 31
8.6.1 Function destruct 31
8.6.2 Function getInfo 31

8.1 Overview

In file `IndexResolver.sol`

8.2 Variable Definitions

TvmCell	_codeIndex	
		used in @1.NftRoot.mintNft
		assigned in @1.NftRoot.:constructor
		used in @1.NftRoot.:constructor
		assigned in @7.Data.:constructor
		used in @7.Data.:constructor
		used in @6.IndexResolver._buildIndexCode

```
    TvmCell _codeIndex;
```

8.3 Public Method Definitions

8.3.1 Function resolveCodeHashIndex

- TODO

```
    function resolveCodeHashIndex(
        address addrRoot,
        address addrOwner
    ) public view returns (uint256 codeHashIndex) {
        return tvm.hash(_buildIndexCode(addrRoot, addrOwner));
    }
```

8.3.2 Function resolveIndex

- TODO

```
    function resolveIndex(
        address addrRoot,
        address addrData,
        address addrOwner
    ) public view returns (address addrIndex) {
        TvmCell code = _buildIndexCode(addrRoot, addrOwner);
        TvmCell state = _buildIndexState(code, addrData);
        uint256 hashState = tvm.hash(state);
        addrIndex = address.makeAddrStd(0, hashState);
    }
```

8.4 Internal Method Definitions

8.4.1 Function _buildIndexCode

- TODO

```
    function _buildIndexCode(
        address addrRoot,
        address addrOwner
    ) internal virtual view returns (TvmCell) {
        TvmBuilder salt;
        salt.store(addrRoot);
        salt.store(addrOwner);
        return tvm.setCodeSalt(_codeIndex, salt.toCell());
    }
```

8.4.2 Function _buildIndexState

- TODO

```
    function _buildIndexState(
        TvmCell code,
        address addrData
    ) internal virtual pure returns (TvmCell) {
        return tvm.buildStateInit({
            contr: Index,
            varInit: {_addrData: addrData},
            code: code
        });
    }
```

Chapter 9

Contract NftRoot

Contents

9.1 Overview 32
9.2 Static Variable Definitions 32
9.3 Modifier Definitions 32
9.3.1 Modifier onlyRoot 32
9.4 Constructor Definitions 33
9.4.1 Constructor 33
9.5 Public Method Definitions 33
9.5.1 Function destruct 33
9.5.2 Function getInfo 33

9.1 Overview

In file `NftRoot.sol`

9.2 Contract Inheritance

DataResolver	
IndexResolver	

9.3 Variable Definitions

uint256	_totalMinted	
		assigned in @1.NftRoot.mintNft
		used in @1.NftRoot.mintNft
		used in @1.NftRoot.mintNft
address	_addrBasis	
		used in @1.NftRoot.destructBasis
		assigned in @1.NftRoot.deployBasis
		used in @1.NftRoot.deployBasis

```
    uint256 _totalMinted;
```

```
    address _addrBasis;
```

Major issue: No way to get funds back
Tokens on the contract are locked forever. This happens when IndexBasis are destroyed and when contracts are deployed.

9.4 Constructor Definitions

9.4.1 Constructor

Minor issue: Variable initialization
The globals _totalMinted and _addrBasis are not initialized.

Minor issue: Code initialization
Anyone can build a NftRoot contract with a fake _codeData and _codeIndex ; consider checking the contract hash.

```
    constructor(TvmCell codeIndex, TvmCell codeData) public {
        tvm.accept();
        _codeIndex = codeIndex;
        _codeData = codeData;
    }
```

9.5 Public Method Definitions

9.5.1 Function deployBasis

Minor issue: Constants
Values "0.5 ton", "0.4 ton" and "104" should be defined as constants.

Minor issue: Variable name typo
Variable "codeHasData" should be named "codeHashData".

- TODO

Minor issue: Unclear behavior

_addrBasis is updated after a call of deployBasis, hence a call of this function forbids the deletion of the previous IndexBasis deployed.

```
    function deployBasis(TvmCell codeIndexBasis) public {
        require(msg.value > 0.5 ton, 104);
        uint256 codeHasData = resolveCodeHashData();
        TvmCell state = tvm.buildStateInit({
            contr: IndexBasis,
            varInit: {
                _codeHashData: codeHasData,
                _addrRoot: address(this)
            },
            code: codeIndexBasis
        });
        _addrBasis = new IndexBasis{stateInit: state, value: 0.4
            ton}();
    }
```

9.5.2 Function destructBasis

Critical issue: Public visibility

This function can be called by anyone ; the authentification of destruct in IndexBasis is useless.

```
    function destructBasis() public view {
        IIndexBasis(_addrBasis).destruct();
    }
```

9.5.3 Function mintNft

Minor issue: Constants

Value "1.1 ton" should be defined a constants.

Minor issue: Spurious variable name

_totalMinted does not represent the total of NFT minted, as the contract creation may bounce.

Major issue: Sender may be null

If a user calls "mintNft", a Data contract is deployed with a null owner.

```
    function mintNft() public {
        TvmCell codeData = _buildDataCode(address(this));
        TvmCell stateData = _buildDataState(codeData, _totalMinted)
            ;
        new Data{stateInit: stateData, value: 1.1 ton}(msg.sender,
            _codeIndex);

        _totalMinted++;
    }
```