Audit

By OCamlPro

September 7, 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

Library Modules

4.1 Module "Constants.sol"

4.1.1 Pragmas

ton	-solidity >= 0.43.0	

4.1.2 Contract Definitions

- Constants

4.2 Module "true_nft_audit.sol"

4.2.1 Imports

../components/true-nft-core/src/NftRoot.sol	

Chapter 5

Interface Modules

5.1 Module "IData.sol"

5.1.1 Pragmas

ton	-solidity >= 0.43.0	

5.1.2 Contract Definitions

- IData

5.2 Module "IIndex.sol"

5.2.1 Pragmas

ton	-solidity >= 0.43.0	

5.2.2 Contract Definitions

- IIndex

5.3 Module "IIndexBasis.sol"

5.3.1 Pragmas

ton	-solidity >= 0.43.0	

5.3.2 Contract Definitions

- IIndexBasis

Chapter 6

Contract Modules

6.1 Module "Data.sol"

6.1.1 Pragmas

ton	-solidity >=0.43.0	
AbiHeader	expire	
AbiHeader	time	

6.1.2 Imports

./resolvers/IndexResolver.sol	
./interfaces/IData.sol	
./libraries/Constants.sol	

6.1.3 Contract Definitions

- Data

6.2 Module "DataResolver.sol"

6.2.1 Pragmas

ton	-solidity >= 0.43.0	
AbiHeader	expire	
AbiHeader	time	

6.2.2 Imports

../Data.sol	

6.2.3 Contract Definitions

- DataResolver

6.3 Module "Index.sol"

6.3.1 Pragmas

ton	-solidity >=0.43.0	
AbiHeader	expire	
AbiHeader	time	

6.3.2 Imports

./interfaces/IIndex.sol	

6.3.3 Contract Definitions

- Index

6.4 Module "IndexBasis.sol"

6.4.1 Pragmas

ton	-solidity >=0.43.0	
AbiHeader	expire	
AbiHeader	time	

6.4.2 Contract Definitions

- IndexBasis

6.5 Module "IndexResolver.sol"

6.5.1 Pragmas

ton	-solidity >= 0.43.0	
AbiHeader	expire	
AbiHeader	time	

6.5.2 Imports

../Index.sol	

6.5.3 Contract Definitions

- IndexResolver

6.6 Module "NftRoot.sol"

6.6.1 Pragmas

ton	-solidity >=0.43.0	
AbiHeader	expire	
AbiHeader	time	

6.6.2 Imports

./resolvers/IndexResolver.sol	
./resolvers/DataResolver.sol	
./IndexBasis.sol	
./interfaces/IData.sol	
./interfaces/IIndexBasis.sol	

6.6.3 Contract Definitions

- NftRoot

Chapter 7

Contract Data

Contents

7.1 Overview

In file `Data.sol`

7.2 Contract Inheritance

IData	
IndexResolver	

7.3 Static Variable Definitions

uint256	_id	

```
18     uint256 static _id;
```

7.4 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

```
14     address _addrRoot;
15     address _addrOwner;
16     address _addrAuthor;
```

7.5 Constructor Definitions

7.5.1 Constructor

Critical issue: Constructor for Data (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum
loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    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);
    }
```

7.6 Public Method Definitions

7.6.1 Function getInfo

- TODO

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

7.6.2 Function getOwner

- TODO

```
69      function getOwner() public view override returns(address
            addrOwner) {
70          addrOwner = _addrOwner;
71      }
```

7.6.3 Function transferOwnership

- TODO

```
35      function transferOwnership(address addrTo) public override {
36          require(msg.sender == _addrOwner);
37          require(msg.value >= Constants.MIN_FOR_DEPLOY);
38
39          address oldIndexOwner = resolveIndex(_addrRoot, address(
                this), _addrOwner);
40          IIndex(oldIndexOwner).destruct();
41          address oldIndexOwnerRoot = resolveIndex(address(0),
                address(this), _addrOwner);
42          IIndex(oldIndexOwnerRoot).destruct();
43
44          _addrOwner = addrTo;
45
46          deployIndex(addrTo);
47      }
```

7.7 Internal Method Definitions

7.7.1 Function deployIndex

- TODO

```
49      function deployIndex(address owner) private {
50          TvmCell codeIndexOwner = _buildIndexCode(_addrRoot, owner);
51          TvmCell stateIndexOwner = _buildIndexState(codeIndexOwner,
                address(this));
52          new Index{stateInit: stateIndexOwner, value: 0.4 ton}(
                _addrRoot);
53
54          TvmCell codeIndexOwnerRoot = _buildIndexCode(address(0),
                owner);
55          TvmCell stateIndexOwnerRoot = _buildIndexState(
                codeIndexOwnerRoot, address(this));
56          new Index{stateInit: stateIndexOwnerRoot, value: 0.4 ton}(
                _addrRoot);
57      }
```

Chapter 8

Contract DataResolver

Contents

8.1 Overview

In file `DataResolver.sol`

8.2 Variable Definitions

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

```
11     TvmCell _codeData;
```

8.3 Public Method Definitions

8.3.1 Function resolveCodeHashData

- TODO

```
13      function resolveCodeHashData() public view returns (uint256
            codeHashData) {
14          return tvm.hash(_buildDataCode(address(this)));
15      }
```

8.3.2 Function resolveData

- TODO

```
17      function resolveData(
18          address addrRoot,
19          uint256 id
20      ) public view returns (address addrData) {
21          TvmCell code = _buildDataCode(addrRoot);
22          TvmCell state = _buildDataState(code, id);
23          uint256 hashState = tvm.hash(state);
24          addrData = address.makeAddrStd(0, hashState);
25      }
```

8.4 Internal Method Definitions

8.4.1 Function _buildDataCode

- TODO

```
27      function _buildDataCode(address addrRoot) internal virtual view
            returns (TvmCell) {
28          TvmBuilder salt;
29          salt.store(addrRoot);
30          return tvm.setCodeSalt(_codeData, salt.toCell());
31      }
```

8.4.2 Function _buildDataState

- TODO

```
33      function _buildDataState(
34          TvmCell code,
35          uint256 id
36      ) internal virtual pure returns (TvmCell) {
37          return tvm.buildStateInit({
38              contr: Data,
```

```
39              varInit: {_id: id},
40              code: code
41          });
42      }
```

Chapter 9

Contract Index

Contents

9.1 Overview

In file `Index.sol`

9.2 Contract Inheritance

IIndex	

9.3 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;
```

9.4 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;
```

9.5 Constructor Definitions

9.5.1 Constructor

Critical issue: Constructor for Index (fake)

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum loren ipsum

- TODO

```
    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;
```

```
24          if(addrRoot == address(0)) {
25              _addrRoot = root;
26          }
27      }
```

9.6 Public Method Definitions

9.6.1 Function destruct

- TODO

```
39      function destruct() public override {
40          require(msg.sender == _addrData);
41          selfdestruct(_addrData);
42      }
```

9.6.2 Function getInfo

- TODO

```
29      function getInfo() public view override returns (
30          address addrRoot,
31          address addrOwner,
32          address addrData
33      ) {
34          addrRoot = _addrRoot;
35          addrOwner = _addrOwner;
36          addrData = _addrData;
37      }
```

Chapter 10

Contract IndexBasis

Contents

10.1 Overview

In file `IndexBasis.sol`

10.2 Static Variable Definitions

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

```
7     address static _addrRoot;

8     uint256 static _codeHashData;
```

10.3 Modifier Definitions

10.3.1 Modifier onlyRoot

```
10      modifier onlyRoot() {
11          require(msg.sender == _addrRoot, 100);
12          tvm.accept();
13          _;
14      }
```

10.4 Constructor Definitions

10.4.1 Constructor

Critical issue: Constructor for IndexBasis (fake)

loren ipsum loren ipsum

- TODO

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

10.5 Public Method Definitions

10.5.1 Function destruct

- TODO

```
23      function destruct() public onlyRoot {
24          selfdestruct(_addrRoot);
25      }
```

10.5.2 Function getInfo

- TODO

```
18      function getInfo() public view returns (address addrRoot,
            uint256 codeHashData) {
19          addrRoot = _addrRoot;
20          codeHashData = _codeHashData;
21      }
```

Chapter 11

Contract IndexResolver

Contents

11.1 Overview

In file `IndexResolver.sol`

11.2 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

```
11      TvmCell _codeIndex;
```

11.3 Public Method Definitions

11.3.1 Function resolveCodeHashIndex

- TODO

```
13      function resolveCodeHashIndex(
14          address addrRoot,
15          address addrOwner
16      ) public view returns (uint256 codeHashIndex) {
17          return tvm.hash(_buildIndexCode(addrRoot, addrOwner));
18      }
```

11.3.2 Function resolveIndex

- TODO

```
20      function resolveIndex(
21          address addrRoot,
22          address addrData,
23          address addrOwner
24      ) public view returns (address addrIndex) {
25          TvmCell code = _buildIndexCode(addrRoot, addrOwner);
26          TvmCell state = _buildIndexState(code, addrData);
27          uint256 hashState = tvm.hash(state);
28          addrIndex = address.makeAddrStd(0, hashState);
29      }
```

11.4 Internal Method Definitions

11.4.1 Function _buildIndexCode

- TODO

```
31      function _buildIndexCode(
32          address addrRoot,
33          address addrOwner
34      ) internal virtual view returns (TvmCell) {
35          TvmBuilder salt;
36          salt.store(addrRoot);
37          salt.store(addrOwner);
38          return tvm.setCodeSalt(_codeIndex, salt.toCell());
39      }
```

11.4.2 Function _buildIndexState

- TODO

```
41      function _buildIndexState(
42          TvmCell code,
43          address addrData
44      ) internal virtual pure returns (TvmCell) {
45          return tvm.buildStateInit({
46              contr: Index,
47              varInit: {_addrData: addrData},
48              code: code
49          });
50      }
```

Chapter 12

Contract NftRoot

Contents

12.1 Overview

In file `NftRoot.sol`

12.2 Contract Inheritance

DataResolver	
IndexResolver	

12.3 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;
```

12.4 Constructor Definitions

12.4.1 Constructor

Critical issue: Constructor for NftRoot (fake)
loren ipsum loren ipsum

- TODO

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

12.5 Public Method Definitions

12.5.1 Function deployBasis

- TODO

```
    function deployBasis(TvmCell codeIndexBasis) public {
        require(msg.value > 0.5 ton, 104);
        uint256 codeHasData = resolveCodeHashData();
        TvmCell state = tvm.buildStateInit({
```

```
37              contr: IndexBasis,
38              varInit: {
39                  _codeHashData: codeHasData,
40                  _addrRoot: address(this)
41              },
42              code: codeIndexBasis
43          });
44          _addrBasis = new IndexBasis{stateInit: state, value: 0.4
                ton}();
45      }
```

12.5.2 Function destructBasis

- TODO

```
47      function destructBasis() public view {
48          IIndexBasis(_addrBasis).destruct();
49      }
```

12.5.3 Function mintNft

- TODO

```
25      function mintNft() public {
26          TvmCell codeData = _buildDataCode(address(this));
27          TvmCell stateData = _buildDataState(codeData, _totalMinted)
                ;
28          new Data{stateInit: stateData, value: 1.1 ton}(msg.sender,
                _codeIndex);
29
30          _totalMinted++;
31      }
```