

Audit

By OCamlPro

September 10, 2021

Table of Minor, Major and Critical Issues

Major issue: Funds accessibility and bounced messages	5
Minor issue: Unsafe assumptions of message origin	5
Minor issue: Naming convention	5
Major issue: No way to get funds back	7
Minor issue: Constants	9
Major issue: addrOwner may be null	9
Critical issue: Methods called without value TODO: CHECK	9
Major issue: New owner may be null	9
Minor issue: New owner may be equal to the old one	10
Minor issue: Constants	10
Minor issue: Constants	14
Minor issue: Double initialization of	14
Minor issue: Modifiers	17
Minor issue: Constants	17
Minor issue: Superfluous tvn.accept()	17
Major issue: No way to get funds back	20
Minor issue: Variable initialization	21
Minor issue: Code initialization	21
Minor issue: Constants	21
Minor issue: Variable name typo	21
Minor issue: Unclear behavior	22
Critical issue: Public visibility	22
Minor issue: Constants	22
Minor issue: Spurious variable name	22
Major issue: Sender may be null	22

Contents

1	Introduction	4
2	Overview	5
2.1	Specification	5
2.2	Generic issues	5
3	Contract Data	7
3.1	Overview	7
3.2	Contract Inheritance	7
3.3	Static Variable Definitions	8
3.4	Variable Definitions	8
3.5	Constructor Definitions	9
3.5.1	Constructor	9
3.6	Public Method Definitions	9
3.6.1	Function getInfo	9
3.6.2	Function getOwner	9
3.6.3	Function transferOwnership	9
3.7	Internal Method Definitions	10
3.7.1	Function deployIndex	10
4	Contract DataResolver	11
4.1	Overview	11
4.2	Variable Definitions	11
4.3	Public Method Definitions	12
4.3.1	Function resolveCodeHashData	12
4.3.2	Function resolveData	12
4.4	Internal Method Definitions	12
4.4.1	Function _buildDataCode	12
4.4.2	Function _buildDataState	12
5	Contract Index	13
5.1	Overview	13
5.2	Contract Inheritance	13
5.3	Static Variable Definitions	13

5.4	Variable Definitions	14
5.5	Constructor Definitions	14
5.5.1	Constructor	14
5.6	Public Method Definitions	15
5.6.1	Function destruct	15
5.6.2	Function getInfo	15
6	Contract IndexBasis	16
6.1	Overview	16
6.2	Static Variable Definitions	16
6.3	Modifier Definitions	17
6.3.1	Modifier onlyRoot	17
6.4	Constructor Definitions	17
6.4.1	Constructor	17
6.5	Public Method Definitions	17
6.5.1	Function destruct	17
6.5.2	Function getInfo	17
7	Contract IndexResolver	18
7.1	Overview	18
7.2	Variable Definitions	18
7.3	Public Method Definitions	19
7.3.1	Function resolveCodeHashIndex	19
7.3.2	Function resolveIndex	19
7.4	Internal Method Definitions	19
7.4.1	Function _buildIndexCode	19
7.4.2	Function _buildIndexState	19
8	Contract NftRoot	20
8.1	Overview	20
8.2	Contract Inheritance	20
8.3	Variable Definitions	21
8.4	Constructor Definitions	21
8.4.1	Constructor	21
8.5	Public Method Definitions	21
8.5.1	Function deployBasis	21
8.5.2	Function destructBasis	22
8.5.3	Function mintNft	22

Chapter 1

Introduction

This informal audit details the functioning of the TrueNFT smart contracts of the FreeTon repository. The content of true-nft-core contains a simplified version a more complex NFT project; it only contains a high level version synthesizing the storage of the different contracts so as to retrieve them with DeBots (the DeBots code were not part of the audited contracts). This report contains a full review of the code with the different issues, from minor to critical, that were found and fixes are proposed in some cases.

This document is the submission to the 17th contest of the ForMet sub-governance. <https://formet.gov.freeton.org/proposal?proposalAddress=0%3Afad8efb6b82a8b321d8b4ce424c8e>

Chapter 2

Overview

2.1 Specification

Non Fungible Tokens are deployed from a **NftRoot** contract which will be the basis of all minted tokens. This contract has two purposes.

- The deployment of a **Data** contract, representing a fraction of the digitalized asset. So as to easily retrieve the information of the asset, it deploys two **Index** contracts: one retrievable from the NftRoot address, the owner's address and the Data address, and one from the NftRoot address and the owner's only.
- The deployment of an **IndexBasis** contract with the Data code hash.

These four contracts represent the whole NFT core implementation. Many interesting features of this infrastructure comes from the use of salted code for generating specific code hashes that can be found with specific instructions accessible by DeBots. This audit does not discuss the retriability of the contracts, but our study of the TrueNFT functioning did not show critical issues in this regard.

2.2 Generic issues

Before reading in detail the source code, several issues (mostly coding habits) affected the project as a whole. We list them in this section.

Major issue: Funds accessibility and bounced messages
Unless a contract is destroyed (which is not the case for all contracts), funds are not accessible. There is no error handling, especially for bounced messages, which allow funds to be stored on contracts.
Minor issue: Unsafe assumptions of message origin
The messages are assumed to be from a contract ; the case <code>msg.sender == address(0)</code> is never treated.

Minor issue: Naming convention

Static variables should start with a prefix like "s_" and globals should start with a prefix like "g_" or "m_" and internal/private functions should start with "_".

Chapter 3

Contract Data

Contents

3.1 Overview	7
3.2 Contract Inheritance	7
3.3 Static Variable Definitions	8
3.4 Variable Definitions	8
3.5 Constructor Definitions	9
3.5.1 Constructor	9
3.6 Public Method Definitions	9
3.6.1 Function getInfo	9
3.6.2 Function getOwner	9
3.6.3 Function transferOwnership	9
3.7 Internal Method Definitions	10
3.7.1 Function deployIndex	10

3.1 Overview

Major issue: No way to get funds back

Tokens sent to the contract are locked forever. They are sent when IndexBasis are destroyed and when contracts are deployed.

3.2 Contract Inheritance

IData	
IndexResolver	

3.3 Static Variable Definitions

uint256	_id	
---------	-----	--

```
18     uint256 static _id;
```

3.4 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;
```

3.5 Constructor Definitions

3.5.1 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.

```

20     constructor(address addrOwner, TvmCell codeIndex) public {
21         optional(TvmCell) optSalt = tvm.codeSalt(tvm.code());
22         require(optSalt.hasValue(), 101);
23         (address addrRoot) = optSalt.get().toSlice().decode(address
24             );
25         require(msg.sender == addrRoot);
26         require(msg.value >= Constants.MIN_FOR_DEPLOY);
27         tvm.accept();
28         _addrRoot = addrRoot;
29         _addrOwner = addrOwner;
30         _addrAuthor = addrOwner;
31         _codeIndex = codeIndex;
32         deployIndex(addrOwner);
33     }

```

3.6 Public Method Definitions

3.6.1 Function getInfo

```

59     function getInfo() public view override returns (
60         address addrRoot,
61         address addrOwner,
62         address addrData
63     ) {
64         addrRoot = _addrRoot;
65         addrOwner = _addrOwner;
66         addrData = address(this);
67     }

```

3.6.2 Function getOwner

```

69     function getOwner() public view override returns(address
70         addrOwner) {
71         addrOwner = _addrOwner;
72     }

```

3.6.3 Function transferOwnership

Critical issue: Methods called without value TODO: CHECK

Index(_).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.

```

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(
40             this), _addrOwner);
41         IIndex(oldIndexOwner).destruct();
42         address oldIndexOwnerRoot = resolveIndex(address(0),
43             address(this), _addrOwner);
44         IIndex(oldIndexOwnerRoot).destruct();
45
46         _addrOwner = addrTo;
47         deployIndex(addrTo);
48     }

```

3.7 Internal Method Definitions

3.7.1 Function deployIndex

Minor issue: Constants

Value "0.4 ton" should be defined as a constant

```

49     function deployIndex(address owner) private {
50         TvmCell codeIndexOwner = _buildIndexCode(_addrRoot, owner);
51         TvmCell stateIndexOwner = _buildIndexState(codeIndexOwner,
52             address(this));
53         new Index{stateInit: stateIndexOwner, value: 0.4 ton}(_
54             _addrRoot);
55
56         TvmCell codeIndexOwnerRoot = _buildIndexCode(address(0),
57             owner);
58         TvmCell stateIndexOwnerRoot = _buildIndexState(
59             codeIndexOwnerRoot, address(this));
60         new Index{stateInit: stateIndexOwnerRoot, value: 0.4 ton}(
61             _addrRoot);
62     }

```

Chapter 4

Contract DataResolver

Contents

4.1 Overview	11
4.2 Variable Definitions	11
4.3 Public Method Definitions	12
4.3.1 Function resolveCodeHashData	12
4.3.2 Function resolveData	12
4.4 Internal Method Definitions	12
4.4.1 Function _buildDataCode	12
4.4.2 Function _buildDataState	12

4.1 Overview

In file `DataResolver.sol`

4.2 Variable Definitions

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

11 TvmCell _codeData;

4.3 Public Method Definitions

4.3.1 Function resolveCodeHashData

```

13     function resolveCodeHashData() public view returns (uint256
14         codeHashData) {
15         return tvn.hash(_buildDataCode(address(this)));
16     }

```

4.3.2 Function resolveData

```

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 = tvn.hash(state);
24         addrData = address.makeAddrStd(0, hashState);
25     }

```

4.4 Internal Method Definitions

4.4.1 Function _buildDataCode

```

27     function _buildDataCode(address addrRoot) internal virtual view
28         returns (TvmCell) {
29         TvmBuilder salt;
30         salt.store(addrRoot);
31         return tvn.setCodeSalt(_codeData, salt.toCell());
32     }

```

4.4.2 Function _buildDataState

```

33     function _buildDataState(
34         TvmCell code,
35         uint256 id
36     ) internal virtual pure returns (TvmCell) {
37         return tvn.buildStateInit({
38             contr: Data,
39             varInit: {_id: id},
40             code: code
41         });
42     }

```

Chapter 5

Contract Index

Contents

5.1	Overview	13
5.2	Contract Inheritance	13
5.3	Static Variable Definitions	13
5.4	Variable Definitions	14
5.5	Constructor Definitions	14
5.5.1	Constructor	14
5.6	Public Method Definitions	15
5.6.1	Function destruct	15
5.6.2	Function getInfo	15

5.1 Overview

In file `Index.sol`

5.2 Contract Inheritance

IIndex	
--------	--

5.3 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

```
11 address static _addrData;
```

5.4 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

```
9 address _addrRoot;
```

```
10 address _addrOwner;
```

5.5 Constructor Definitions

5.5.1 Constructor

Minor issue: Constants

Values "101" and "address(0)" should be constants.

Minor issue: Double initialization of

_addrRoot is initialized twice if addrRoot = 0.

```
13 constructor(address root) public {
14     optional(TvmCell) optSalt = tvn.codeSalt(tvn.code());
15     require(optSalt.hasValue(), 101);
16     (address addrRoot, address addrOwner) = optSalt
17         .get()
18         .toSlice()
19         .decode(address, address);
20     require(msg.sender == _addrData);
21     tvn.accept();
22     _addrRoot = addrRoot;
23     _addrOwner = addrOwner;
24     if(addrRoot == address(0)) {
25         _addrRoot = root;
26     }
27 }
```

5.6 Public Method Definitions

5.6.1 Function destruct

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

5.6.2 Function getInfo

```
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 6

Contract IndexBasis

Contents

6.1 Overview	16
6.2 Static Variable Definitions	16
6.3 Modifier Definitions	17
6.3.1 Modifier onlyRoot	17
6.4 Constructor Definitions	17
6.4.1 Constructor	17
6.5 Public Method Definitions	17
6.5.1 Function destruct	17
6.5.2 Function getInfo	17

6.1 Overview

In file `IndexBasis.sol`

6.2 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;
```

6.3 Modifier Definitions

6.3.1 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.

```

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

6.4 Constructor Definitions

6.4.1 Constructor

```

16     constructor() public onlyRoot {}
```

6.5 Public Method Definitions

6.5.1 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).

```

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

6.5.2 Function getInfo

```

18     function getInfo() public view returns (address addrRoot,
19         uint256 codeHashData) {
20         addrRoot = _addrRoot;
21         codeHashData = _codeHashData;
22     }
```

Chapter 7

Contract IndexResolver

Contents

7.1	Overview	18
7.2	Variable Definitions	18
7.3	Public Method Definitions	19
7.3.1	Function resolveCodeHashIndex	19
7.3.2	Function resolveIndex	19
7.4	Internal Method Definitions	19
7.4.1	Function _buildIndexCode	19
7.4.2	Function _buildIndexState	19

7.1 Overview

In file `IndexResolver.sol`

7.2 Variable Definitions

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

11 `TvmCell _codeIndex;`

7.3 Public Method Definitions

7.3.1 Function resolveCodeHashIndex

```

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

```

7.3.2 Function resolveIndex

```

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     }

```

7.4 Internal Method Definitions

7.4.1 Function _buildIndexCode

```

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     }

```

7.4.2 Function _buildIndexState

```

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 8

Contract NftRoot

Contents

8.1	Overview	20
8.2	Contract Inheritance	20
8.3	Variable Definitions	21
8.4	Constructor Definitions	21
8.4.1	Constructor	21
8.5	Public Method Definitions	21
8.5.1	Function deployBasis	21
8.5.2	Function destructBasis	22
8.5.3	Function mintNft	22

8.1 Overview

Major issue: No way to get funds back

Tokens sent to the contract are locked forever. They are sent when IndexBasis are destroyed and when contracts are deployed.

8.2 Contract Inheritance

DataResolver	
IndexResolver	

8.3 Variable Definitions

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

```
16  uint256 _totalMinted;
```

```
17  address _addrBasis;
```

8.4 Constructor Definitions

8.4.1 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.

```
19  constructor(TvmCell codeIndex, TvmCell codeData) public {
20      tvm.accept();
21      _codeIndex = codeIndex;
22      _codeData = codeData;
23  }
```

8.5 Public Method Definitions

8.5.1 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.

```

33     function deployBasis(TvmCell codeIndexBasis) public {
34         require(msg.value > 0.5 ton, 104);
35         uint256 codeHasData = resolveCodeHashData();
36         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
45             ton}();

```

8.5.2 Function destructBasis**Critical issue: Public visibility**

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

```

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

```

8.5.3 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.

```

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

```