SmartDec

NaviToken Smart Contracts Security Analysis

This report is public.

Published: February 28, 2018

Abstract

In this report we consider the security of the NaviToken project. Our task is to find and describe security issues in the smart contracts of the project.

Procedure

In our audit, we consider the following crucial features of the smart contract code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices in efficient use of gas, code readability, etc.

We perform our audit according to the following procedure:

- automated analysis
  - we scan the smart contracts with our own Solidity static code analyzer SmartCheck
  - we scan the smart contracts with several publicly available automated Solidity analysis tools such as Remix, Solhint, Oyente and Securify (beta version since full version was unavailable at the moment this report was made)
  - we manually verify (reject or confirm) the issues found by tools
- manual audit
  - we manually analyze the smart contracts for security vulnerabilities
  - we check the smart contracts logic and compare it with the one described in the documentation
  - we check ERC20 compliance
  - we run tests and check code coverage
- report
  - we reflect all the gathered information in the report

Disclaimer

The audit does not give any warranties on the security of the code. One audit can not be considered enough. We always recommend proceeding to several independent audits and a public bug bounty program to ensure the security of the smart contracts. Besides, security audit is not an investment advice.

Checked vulnerabilities

We have scanned the NaviToken smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DOS with (Unexpected) revert
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Project Overview

In our analysis we consider NaviToken whitepaper (version on 2017-12-28_v2) and smart contracts code (version on commit 9ab694e).

Project Architecture

For the audit, we have been provided with the following set of files:

- NaviTokem.sol inherits the Ownable and StandardToken contracts from the OpenZeppelin library of version ^1.5.0.
  At the time of the audit up-to-date version is 1.7.0.
- Migrations.sol

The files are the part of the truffle project. The project also contains tests, deploy scripts, and several files that are beyond the scope of the audit.

The total volume of the Solidity code that has been audited is 102 lines of code.
The project compiles successfully with `truffle compile` command (see the Compilation output in the Appendix)
The project successfully passes the tests (`truffle test` command, tests output).

Code Logic

The NaviToken contract implements the ERC20 token logic (StandardToken contract from OpenZeppelin library of version 1.5.0) with some modifications and violations described below. The token parameters:

- name: “NaviToken”
- symbol: “NVT”
- decimals: 18
- total supply: 1,000,000,000

The NaviToken contract is ownable (Ownable from OpenZeppelin library of version 1.5.0).

The owner of the contract is the address from which the contract is deployed. When the contract is deployed, 10% of tokens are transferred to the owner's address, while other tokens are not assigned to anyone.

The owner of the contract could call the `batchAssignTokens` function to add address for future defrost of tokens until the `stopBatchAssign` function is called by the owner. Addresses could be of three kinds: investors, reserve and team, and advisors. Each kind has different requirements of defrosting.

All investors receive tokens immediately during call of the `batchAssignTokens` function.
All reserve and team addresses receive tokens after calling `defrostReserveAndTeamTokens` function by owner after `DEFROST_AFTER_MONTHS` (set to 6 month) after ICO starts `START_ICO_TIMESTAMP`. Desired address receives its amount of tokens limited to current possible unfrosted amount equal to `1/DEFROST_FACTOR_TEAMANDADV` (30 parts) part each month from defrosting time.
All Advisors receive tokens after calling the defrostAdvisorsTokens function by owner after `DEFROST_AFTER_MONTHS` (set to 6 month) after ICO starts `START_ICO_TIMESTAMP`.

The owner of the contract can call `stopBatchAssign` any time to stop the `batchAssignTokens` function.

The NaviToken smart contract does not cover process of receiving ETH or any other currency. Neither it covers token distribution according to the amount received during the tokensale.

Automated Analysis

We used several publicly available automated Solidity analysis tools.
Here are the combined results of SmartCheck, Solhint, and Remix. Securify (beta-version) and Oyente have found no issues.
All the issues found by tools were manually checked (rejected or confirmed).

Tool	Rule	False positive	True positive
SmartCheck	Dos with revert	1	
	Gas limit and loops		3
	No payable fallback	2	
	Pragmas version		2
	Reentrancy external call	3	
	Timestamp dependence	1	
	Unchecked math	5	
	Visibility	5	
Total SmartCheck		17	5
Remix	Gas requirement of function high	11	3
	use of "now"		5
	Variables have very similar names	2	
Total Remix		18	3
Solhint	Avoid to make time-based decisions in your business logic		5

Compiler version must be fixed		1
Explicitly mark visibility of state		5
Total Solhint		11
Total Overall	**30**	**24**

Securify* — beta version, full version is unavailable.

Cases when these issues lead to actual bugs or vulnerabilities are described in the next section.

Manual Analysis

The contracts were completely manually analyzed, their logic was checked and compared with the one described in the documentation. Besides, the results of the automated analysis were manually verified. All confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

Gas limit and loops

Consider tokens distribution via the `batchAssignTokens` function. If owner adds more than 120-150 addresses of the `ReserveAndTeam` type, or more than 200 addresses of the `Advisor` type, the `owner` will not be able to unfrost token by calling `defrostReserveAndTeamTokens` and `defrostAdvisorsTokens` respectively. This will happen due to gas limit of block. We recommend rewriting logic of defrosting or adding batch defrosting functionality.
See test example in Appendix.

Medium severity issues

Medium issues can influence smart contracts operation in current implementation. We highly recommend addressing them.

Modularity

It is undesirable to implement both token and non-trivial logic in one contract. If issues are found in logic after deploy and the developer has to update the contract, he/she might encounter migration problems with NVT token.
We highly recommend making the system more modular. In this particular case, we recommend implementing two separate contracts: one of them should implement the token, and the other should implement the platform logic.

Low severity issues

Low severity issues can influence smart contracts operation in future versions of code. We recommend to take them into account.

Non-release code

The contract code contains code that must be removed or changed before deployment for production use.

- NaviToken.sol, line 22

```
uint256 public START_ICO_TIMESTAMP;
```

- NaviToken.sol, line 54

```
START_ICO_TIMESTAMP = now;
```

We highly recommend not to use temporary code for contracts and use constructor parameters and proper deploy scripts.

Discrepancies with the whitepaper

The audit showed the discrepancy within the whitepaper. According to the whitepaper, section 5, 1,000,000,000 NVT tokens will be created during the tokensale and distributed accordingly:

```
"Of the Company NVT, 1,000,000,000 NVT will be issued in
connection with the deployment and the development of the
Platform, of which:
- 500,000,000 NVT will be allocated for distribution during
sale procedures as per Section 6 below;
- 200,000,000 NVT will be allocated to Platform Growth Fund,
which we will administer to incentivize use of the Platform;
- 200,000,000 NVT will be retained by the Company (*subject to
NVT lock-up restrictions); and
- 100,000,000 NVT will be allocated to the Company's
management team and advisors (*subject to NVT lock-up
restrictions)."
```

However, the contract only have distribution limitations for 100,000,000 NVT token (NaviToken.sol, line 48)

```
      uint256 amountReserve    =
MAX_NUM_NAVITOKENS.mul(10).div(100);
```

ERC20 standard violation

The audit showed some deviations from the ERC20 specifications.

1. During the contract creation, constructor should emit the Transfer event with _from set to 0x0 (https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md#transfer-1)
2. In the `batchAssignTokens`, `defrostReserveAndTeamTokens`, `defrostAdvisorsTokens` functions, token transfer or token minting should emit the `Transfer` event.
3. The decimals constant should be `uint8`.

We recommend minting all tokens during the contract construction and then using the `transfer` function of StandardToken in other places without accessing internal balances.

OpenZeppelin version

The package.json does not specify the exact version of the OpenZeppelin library (line 29):

```
"zeppelin-solidity": "^1.5.0"
```

Thus, with the release of version 1.6.0 (it came out at the time of the audit) a new version will be used. But incompatible changes can occur.
For example, in version 1.6.0 in comparison with version 1.5.0, the location of the ERC20 token related files has changed, so the project ceased building successfully.
We recommend specifying the exact versions of the OpenZeppelin library in package.json.

Pragmas version

Solidity source files indicate the versions of the compiler they can be compiled with.
Example:

```
pragma solidity ^0.4.18; // bad: compiles w 0.4.18 and above
pragma solidity 0.4.18; // good: compiles w 0.4.18 only
```

We recommend following the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. Besides, we recommend using the latest compiler version (0.4.20 at the moment). However, these issues were found not in the main contracts, but in OpenZeppelin library.

Conclusion

In this report we have considered the security of NaviToken smart contracts. We performed our audit according to the Procedure described above.
The audit showed one critical issue, one medium, as well as five lower issues. We highly recommend addressing them.

This analysis was performed by SmartDec.

Pavel Yuschenko, Chief Technical Officer
Elizaveta Kharlamova, Analyst
Alexander Seleznev, Chief Business Development Officer

COO Sergey Pavlin 

February 28, 2018

Appendix

Lines of code (cloc tool output)

```
       1 text file.
       1 unique file.
       0 files ignored.

github.com/AlDanial/cloc v 1.74  T=0.00 s (219.6 files/s,
32937.8 lines/s)
-------------------------------------------------------------
-----------------
Language                     files          blank        comme
nt           code
-------------------------------------------------------------
-----------------
Solidity                         1             31
17            102
-------------------------------------------------------------
-----------------
```

Compilation output

```
Compiling ./contracts/Migrations.sol...
Compiling ./contracts/NaviToken.sol...
Compiling zeppelin-solidity/contracts/math/SafeMath.sol...
Compiling zeppelin-solidity/contracts/ownership/Ownable.sol...
Compiling zeppelin-solidity/contracts/token/BasicToken.sol...
Compiling zeppelin-solidity/contracts/token/ERC20.sol...
Compiling zeppelin-solidity/contracts/token/ERC20Basic.sol...
Compiling zeppelin-
solidity/contracts/token/StandardToken.sol...
Writing artifacts to ./build/contracts
```

Tests output

```
Contract: NaviToken
Max Num Navi Tokens = 1e+27
    ✓ should retrieve max number of NaviToken
balance accounts[0] = 1e+26
```

```
    ✓ should retrieve NaviToken owner balance (account[0])
startICOts = 1519479835
    ✓ should retrieve start ICO timestamp
canDefrostRT = false
    ✓ should ask if it can defrost reserve and team and get
answer no (false)
canDefrostAdv = false
    ✓ should ask if it can defrost advisors and get answer no
(false)
elapsedMonths from ICO start = 0
    ✓ should get number of elapsed months from ICO start
    ✓ should assign tokens to three address (one investor, one
team&reserve, one advisor) (72ms)
investor amount = 1.5e+22
    ✓ should retrieve investor assigned tokens amount
canDefrostReserveAndTeam after 7 month = true
    ✓ should can defrost after defrostTime (165ms)
team amount = 2e+22
elapsedMonths from ICO start = 36
    ✓ should assign team token balance after defrost (129ms)
advisor amount = 2.5e+22
    ✓ should assign advisor token balance after defrost (53ms)

  Contract: NaviToken
    ✓ #1 defrost advisor's tokens (139ms)
    ✓ #2 defrost several times in one month (191ms)
    ✓ #3 distribution lifecycle (1417ms)
    ✓ #4 defrost, transfer, defrost (195ms)
    ✓ #5 distribute more tokens than MAX_NUM_NAVITOKENS (84ms)
    ✓ #6 check totalSupply (96ms)
    ✓ #7 double assignment (112ms)

  18 passing (3s)
```

Solium output

```
$ solium --dir contracts/

contracts/NaviToken.sol
```

```
3:7 error "zeppelin-
solidity/contracts/token/StandardToken.sol": Import statements
must use double quotes only. quotes
4:7 error "zeppelin-solidity/contracts/ownership/Ownable.sol":
Import statements must use double quotes only. quotes
5:7 error "zeppelin-solidity/contracts/math/SafeMath.sol":
Import statements must use double quotes only. quotes
12:1 error Only use indent of 4 spaces. indentation
48:8 warning Assignment operator must have exactly single
space on both sides of it. operator-whitespace
49:8 warning Assignment operator must have exactly single
space on both sides of it. operator-whitespace
50:8 warning Assignment operator must have exactly single
space on both sides of it. operator-whitespace
54:30 warning Avoid using 'now' (alias to 'block.timestamp').
security/no-block-members
92:7 error Only use indent of 8 spaces. indentation
92:15 warning Avoid using 'now' (alias to 'block.timestamp').
security/no-block-members
92:50 warning Avoid using 'now' (alias to 'block.timestamp').
security/no-block-members
97:0 error Only use indent of 8 spaces. indentation
97:15 warning Avoid using 'now' (alias to 'block.timestamp').
security/no-block-members
129:0 error Only use indent of 8 spaces. indentation
129:15 warning Avoid using 'now' (alias to 'block.timestamp').
security/no-block-members

✖ 7 errors, 8 warnings found.
```

Out of gas test example

```
const BigNumber = web3.BigNumber;

require('chai')
    .use(require("chai-bignumber")(BigNumber))
    .use(require('chai-as-promised'))
    .should();

const NaviToken = artifacts.require("./NaviToken.sol");

const increaseTime = (addSeconds) => new Promise((resolve,
reject) => {
```

```
    web3.currentProvider.sendAsync(
        [{jsonrpc: "2.0", method: "evm_increaseTime", params:
[addSeconds], id: 0},
            {jsonrpc: "2.0", method: "evm_mine", params: [],
id: 0}
        ],
        function (error, result) {
            if (error) {
                reject(error);
            } else {
                resolve(result);
            }
        }
    );
});

const seconds = (amount) => amount;
const minutes = (amount) => amount * seconds(60);
const hours = (amount) => amount * minutes(60);
const days = (amount) => amount * hours(24);
const months = (amount) => amount * days(30);

const TOKEN_DECIMALS_MULTIPLIER = 10**18;

function makePseudoAdress() {
    var text = "";
    var possible = "abcdef0123456789";

    for (var i = 0; i < 40; i++)
        text += possible.charAt(Math.floor(Math.random() *
possible.length));

    return "0x" + text;
}

contract('NaviToken', accounts => {
    describe('ReserveAndTeam', function() {
        it('Defrosting pass with 120 addresses', async () => {
            const token = await NaviToken.new();
            let addresses = [];
            let amounts = [];
            let classes = [];
            for (let i = 0; i < 100; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(1);
            }
```

```
            await token.batchAssignTokens(addresses, amounts,
classes);

            addresses = [];
            amounts = [];
            classes = [];
            for (let i = 0; i < 20; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(1);
            }
            await token.batchAssignTokens(addresses, amounts,
classes);


            await increaseTime(months(7));
            await token.defrostReserveAndTeamTokens();
        });
        it('Defrosting fail with 130 addresses', async () => {
            const token = await NaviToken.new();
            let addresses = [];
            let amounts = [];
            let classes = [];
            for (let i = 0; i < 100; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(1);
            }
            await token.batchAssignTokens(addresses, amounts,
classes);

            addresses = [];
            amounts = [];
            classes = [];
            for (let i = 0; i < 50; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(1);
            }
            await token.batchAssignTokens(addresses, amounts,
classes);

            await increaseTime(months(7));
            await token.defrostReserveAndTeamTokens();
        });
    });
    describe('Advisors', function() {
```

```
        it('Defrosting pass with 200 adresses', async () => {
            const token = await NaviToken.new();
            let addresses = [];
            let amounts = [];
            let classes = [];
            for (let i = 0; i < 100; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(2);
            }
            await token.batchAssignTokens(addresses, amounts,
classes);

            addresses = [];
            amounts = [];
            classes = [];
            for (let i = 0; i < 100; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(2);
            }
            await token.batchAssignTokens(addresses, amounts,
classes);

            await increaseTime(months(7));
            await token.defrostAdvisorsTokens();
        });
        it('Defrosting pass with 250 addresses', async () => {
            const token = await NaviToken.new();
            let addresses = [];
            let amounts = [];
            let classes = [];
            for (let i = 0; i < 100; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(2);
            }
            await token.batchAssignTokens(addresses, amounts,
classes);

            addresses = [];
            amounts = [];
            classes = [];
            for (let i = 0; i < 100; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(2);
```

```
            }
            await token.batchAssignTokens(addresses, amounts,
classes);

            addresses = [];
            amounts = [];
            classes = [];
            for (let i = 0; i < 50; i++) {
                addresses.push(makePseudoAdress());
                amounts.push(1000);
                classes.push(2);
            }
            await token.batchAssignTokens(addresses, amounts,
classes);

            await increaseTime(months(7));
            await token.defrostAdvisorsTokens();
        });
    });
});
```