OCEANICO.IO

NAVIADDRESS

SMART CONTRACTS EXPERTISE — RIGHT WAY TO SUCCESS
Naviaddress Smart Contract Audit

If you have any questions concerning smart contract design and audit, feel free to contact zoia@oceanico.io

Content

Description of the complex of procedures for auditing a smart contract

1. Primary architecture review

- Checking the architecture of the contract.
- Correctness of the code.
- Check for linearity, shortness and self-documentation.
- Static verification and code analysis for validity and the presence of syntactic errors.

2. Comparison of requirements and implementation

- Checking the code of the smart contract for compliance with the requirements of the customer code logic, writing algorithms, matching the initial constant values.
- Identification of potential vulnerabilities

3. Testing according to the requirements

- Control testing of a smart contract for compliance with specified customer requirements.
- Running tests of the properties of the smart contract in test net.

Terms of Reference for the creation of a smart contract

Navi-token Smart Contract

Token Sale Details:

- Token symbol: NVT
- Decimals: 18

NVT SALE

Your purchase of NVT from us during Private Sale, Pre-ICO and/or ICO (as defined below) is final, and there are no refunds or cancellations except as may be required by applicable law or regulation. We reserve the right to refuse or cancel NVT purchase requests at any time in our sole discretion.

You acknowledge and agree that there are risks associated with purchasing NVT, holding NVT, and using NVT in connection with Naviaddress, the Services and/or Platform including those risks detailed in the Exhibits hereto.

If you have any questions regarding these risks, please contact us at ico@naviaddress.com. BY PURCHASING NVT, YOU EXPRESSLY ACKNOWLEDGE AND ASSUME THESE RISKS.

NVT INITIAL ISSUE AND PLATFORM GROWTH FUND

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:

- 400,000,000 NVT will be allocated for distribution during sale procedures as per Section 6 below;

- 300,000,000 NVT will be allocated to Platform Growth Fund, which we will administer to incentivize use of the Platform (of which 200,000,000 NVT will be subject to NVT lock-up restrictions*);

- 200,000,000 NVT will be retained by the Company (*subject to NVT lock-up restrictions); and100,000,000 NVT will be allocated to the Company's management team and advisors (*subject to NVT lock-up restrictions).

* The Pool of total 500,000,000 NVT (the "NVT Pool") is subject to the following lock-up restrictions during 36 months as of the date of NVT Launch:

• 6 months complete lock-up: NVT are gradually released for sale only upon expiration of 6 months following the date of NVT Launch;

• 30 months lock-up graduate release: NVT are released for sale in 30 installments of 1/30th of NVT Pool, each of these installments occurring on the 30th calendar day of each month after the expiry of the 6 months complete lock-up period.

NVT SALE PROCEDURES AND SPECIFICATIONS

Company will create up to 1 billion NVT through a smart contract on the Ethereum platform ("Smart Contract System"). All NVT will be of equal value and functionality.

The Company's sale of NVT for accredited (qualified) purchasers will be in three stages, each beginning on the dates specified below:

- **Stage 1** effective until 28 January 2018 during which up to 100,000,000 NVT will be distributed in lots of minimum 2,000,000 NVT per purchaser (the "Private Sale");

- **Stage 2** effective from 29 January 2018 until 28 February 2018 during which NVT will be distributed in lots of minimum 200,000 and maximum 2,000,000 NVT per purchaser (the "PreICO"); and in case any NVT remains available for distribution 4

- **Stage 3** effective from 1 March 2018 until 31 March 2018 during which NVT will be distributed in lots of minimum 1000 NVT per purchaser (the "ICO"). In case of oversubscription the amount of NVT per purchaser shall be determined on the pro rata basis.

Unless otherwise agreed with the Company, the sale price of 1 NVT is equal to US$0.05.

During the Private Sale and Pre-ICO Phases the NVT may only be purchased by purchasers who are both Accredited Purchasers and Eligible Purchasers.

Accredited Purchasers are purchasers who are:

- "sophisticated persons" meaning a person- (a) regulated by the Cayman Islands Monetary Authority; (b) any of whose securities are listed on recognised securities exchange; or (c) who - (i) by virtue of knowledge and experience in financial and business matters is reasonably to be regarded as capable of evaluating the merits of a

proposed transaction; and (ii) participates in a transaction with a value or in monetary amounts of at least US$100,000 or its equivalent in any other currency, in the case of each single transaction;

- “high net worth persons” meaning (i) an individual whose net worth is at least $800,000 or its equivalent in any other currency; or (ii) any person that has total assets of not less than $4,000,000 or its equivalent in any other currency; or

- a purchaser who, in their jurisdiction of residence, would satisfy a similar definition prescribed by the laws of such jurisdiction.

The purchase price that you pay for NVT is exclusive of all applicable taxes. You are responsible for determining what, if any, taxes apply to your purchase of NVT, including, for example, sales, use, value added, and similar taxes. It is also your responsibility to withhold, collect, report and remit the correct taxes to the appropriate tax authorities. We are not responsible for withholding, collecting, reporting, or remitting any sales, use, value added, or similar tax arising from your purchase of NVT.

NVT can be purchased with Ethereum, Bitcoin and Fiat currency; to receive the NVT you purchase, you must have an Ethereum wallet that supports the ERC20 token standard.

The Company reserves the right to prescribe additional wallet requirements.

To subscribe for NVT you will be required to:

a) agree to these Terms by clicking the “I Agree with the Terms” at http://naviaddress.io ;

b) confirm that you are an Eligible Purchaser (as defined in Exhibit B to these Terms) and during the Private Sale and Pre-ICO, that you are an Accredited Purchaser;

c) agree to provide such further verification of the identity and source of funds, as requested by the Company, before the application can be processed; and

d) send the relevant amount of Ether, Bitcoin or Fiat currency to the Company’s corresponding wallet/account communicated by the Company.

Within 7 business days after expiration of

i) ICO or ii) Pre-ICO (in case all of 500,000,000 NVT will be distributed during the first two stages), the Company will trigger a Smart Contract System, pursuant to which the Smart Contract System will automatically create and promptly deliver the corresponding NVT to the ERC20 wallet address provided by you (the “NVT Launch”).

All NVT not distributed during above three stages will be assigned to the Platform Growth Fund.

You acknowledge and agree that the smart contracts generated by the Smart Contract System will be governed by the laws of the Cayman Islands and subject to the exclusive jurisdiction of the courts of the Cayman Islands.

Additional information about the smart contract was taken from the sources below

https://naviaddress.io/wp-content/uploads/2018/02/2018-02-23_Terms_of_NaviToken_Sale_v2.3.pdf

List of audited files

Github:

https://github.com/naviworld/navi-token

The following 2 .sol / 9 .js files were audited:

- Migrations.sol
- NaviToken.sol
- contracts/NaviToken.sol
- test/TestNaviToken.js
- generateAssignAccounts.js
- AssignTokensScheduler.js
- TryDefrostAdvisors.js
- TryDefrostReserveAndTeam.js
- verrifyOneInvestorBalance.js
- verifyBatchAssignNormalInvestor.js
- verifyBatchAssignDefrosted.js

Review of smart contract #1

Navi-token smart contract review #1

https://github.com/naviworld/navi-token

Important:

1. If the contract is deployed and there are be more than 30 days to start ICO, then there will be problems when calling the function canDefrostReserveAndTeam() when converting numMonths to uint - overflow.

```
function canDefrostReserveAndTeam() constant returns (bool) {
  int numMonths = elapsedMonthsFromICOStart();
  return numMonths >= DEFROST_AFTER_MONTHS &&
                  uint(numMonths) <=
SafeMath.add(uint(DEFROST_AFTER_MONTHS),  DEFROST_FACTOR_
TEAMANDADV);
}
```

   We recommend that you rewrite elapsedMonthsFromICOStart and all the logic associated with this function by the way, so that you do not use int at all, only uint, this will avoid problems with the conversion of data types.

```
function elapsedMonthsFromICOStart() pure returns (uint elapsed) {
    elapsed = (now <= START_ICO_TIMESTAMP) ?
        0 : (now - START_ICO_TIMESTAMP) / 60 / MONTH_IN_MINUTES;
}
```

2. If you know the function you create only allows for external calls, use external instead public. It provides performance benefits and you will save on gas.

3. The owner = msg.sender line in the NaviToken constructor is useless since the same thing is done in the Ownable constructor, which is called automatically before the NaviToken constructor. We recommend to remove it as this action spends extra gas.

4. The action performed by totalSupply = MAX_NUM_NAVITOKENS in the constructor is false. Since totalsupply shows how many tokens are released in total, and after the contract is actually released there will actually be only amountReserve tokens, and after the call to batchAssignTokens totalSupply does not change. We recommend using the solution from MintableToken (https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/MintableToken.sol) instead of the usual balances[currentAddress] assignment of the new value.

5. assignedSupply is not required. We recommend comparing totalSupply with MAX_NUM_NAVITOKENS.

6. During translating tokens, the totalSupply value should not exceed MAX_NUM_NAVITOKENS In batchAssignTokens. We recommend adding an appropriate check.

7. The defrostReserveAndTeamTokens function contains the require (monthsElapsedTeamAndAdv> 0) condition, which does not allow the token to be thawed before 6 months, but only after 7, but in the technical task this moment is described as follows: *"6 months complete lock-up: NVT are released for sale only upon expiration of 6 months following the date of NVT Launch, "30 months lock-up graduate release: NVT are released for sale in monthly installments of 1 / 30th of the granted NVT on the last calendar day of the month starting from the last day of the 7th month following the date of NVT Launch."* Proceeding from this, the possibility to unfreeze tokens should be in 6 months and 29 days (if this month is 30 days) and not after 7 months.

8. In technical task item 5. NVT INITIAL ISSUE AND PLATFORM GROWTH FUND: the same system acts on Advisor tokens as on team tokens (1/30 payout once a month), and in the contract code all tokens are defrosted immediately in the defrostAdvisorsTokens() method.

9. Due to the fact that canDefrostAdvisors checks elapsedMonthsFromICOStart () == DEFROST_AFTER_MONTNS instead of elapsedMonthsFromICOStart ()> = DEFROST_AFTER_MONTNS, defrosting of funds after DEFROST_AFTER_MONTH will be impossible.

10. We draw your attention to the fact that batchAssignTokens does not work correctly when trying to distribute tokens to the addresses to which the tokens have already been distributed.

11. We draw your attention to the fact that TestNaviToken.js: instance. defrostReserveAndTeamTokens() is an asynchronous method whose result must be waited before to start executing another method.

12. The use of random data in tests is not recommended, as this test will not be repeated. We recommend using a randomly assigned seed.

```
function randomValueAmount () {
  var low =  1;
  var high = 100000;
  return parseInt(Math.random() * (high - low) + low)
}
```

Code quality:

1. There is no need to define functions (lagReserveAndTeamDefrost) that return public contract state variables (DEFROST_AFTER_MONTHS), because web3.js allows you to call instance.DEFROST_AFTER_MONTHS() and get the same result that instance.lagReserveAndTeamDefrost() returns.

2. The constant modifier in functions is recommended to limit to pure/view.

3. A variable with a name icedBalancesTeamAndAdv_frosted stores the balances of Team&Reserve, not Team&Advisors. We do not recommend using camelCase together with under_score in the name of a variable.

4. We draw your attention to the fact that the getAddressBalance function is not required since it duplicates the implementation of the standard ERC20 function in StandardToken.

5. We draw your attention to the fact that the function getAddressAndBalance duplicates method balanceOf (ERC20) which returns a balance.

6. We do not recommend using getBlockTimestamp since Blockchain timestamp can be obtained by means of web3 without using a contract.

7. We recommend using enum DefrostClass {Investor, ReserveAndTeam, Advisor} (https://solidity.readthedocs.io/en/develop/types.html#enums) instead of 0, 1, 2 for enumerate the class of users.

8. In canDefrostReserveAndTeam and at the beginning of defrostReserveAndTeam, there is one and the same test, recorded in different ways. We recommend using require (canDefrostReserveAndTeam()) in defrostReserveAndTeam. The same applies to defrostAdvisorTokens.

9. TestNaviToken.js: Tests inside it() must be independent of each other. If one of the tests is removed or replaced, the result of the automatic tests should not be changed. It is recommended to add beforeEach or afterEach, that before each test the rollback is performed in time to the initial state.

10. TestNaviToken.js: We recommend you revise the architecture of the scripts in favor of using async/await instead of Promise.then to simplify the code. This will help to simplify understanding of the code and its maintenance.

11. TestNaviToken.js: The names of some variables do not say anything about the meaning that these variables store (vret).

```
return NaviToken.deployed().then(function(instance) {
  return instance.getAddressAndBalance(vmyaddr[0]);
}).then(function(vret)
```

12. Assignment without announcement is not recommended. It is recommended to declare variables as var, let or const. You should declare variables as const if you do not want to change the state of these variables. This allows you to notice an error in the logic of the program during the code parsing phase of the IDE.

```
investorAddress = vret[0];
investorAmount = vret[1];
```

TestNaviToken.js: We recommend simplifying this code by replacing

```
var vmyaddr = [];
vmyaddr.push('0xffff2828eeee4545dddd0808cccc7777bbbb0000')
vmyaddr.push('0xffff2828eeee4545dddd0808cccc7777bbbb1111')
vmyaddr.push('0xffff2828eeee4545dddd0808cccc7777bbbb2222')
var vmyamount = [];
vmyamount.push(15000)
vmyamount.push(20000)
vmyamount.push(25000)
var vmyclass = [];
vmyclass.push(0)
vmyclass.push(1)
vmyclass.push(2)
```

on

```
const vmyaddr = [
    '0xffff2828eeee4545dddd0808cccc7777bbbb0000',
    '0xffff2828eeee4545dddd0808cccc7777bbbb1111',
    '0xffff2828eeee4545dddd0808cccc7777bbbb2222',
];
const vmyamount = [15000, 20000, 25000];
const vmyclass = [0, 1, 2];
```

Testing

Testing with instructions

https://github.com/naviworld/navi-token/blob/master/README.txt

1. start ganache-cli
2. truffle compile
3. truffle migrate
4. launch generateAssingAccounts.js
5. launch AssignTokensScheduler.js
6. launch verifyBatchAssignNormalInvestor.js
7. launch TryDefrostAdvisors.js
8. use custom script to increase ganache time to 6 month
9. launch TryDefrostAdvisors.js again. Error
10. but if increase time by another 1 month and launch TryDefrostAdvisors.js again, it will be OK.
11. log (https://gist.github.com/kolya-t/9ad17ca99a249c14745ea715afb2ed15)

Automated tests

https://github.com/naviworld/navi-token/blob/master/README.txt

1. **FAILURE** The check that advisor tokens are not thawed all at once, and by 1/30 every month for 30 months, as reserveAndTeam tokens (Important.8)

2. **SUCCESS** Performing the defrosting function several times in one month with subsequent verification showing that only the first defrost was triggered.

3. **SUCCESS** The Scenario distribution of tokens, their defrosting for 30 months. The Verification shows that all tokens are defrosted correctly: at the specified time and in the right amount.

4. **SUCCESS** Defrost the tokens for the ReserveAndTeam address, then transfer from this token address and again defrost. The check show that the number of defrosted tokens does not depend on the current balance on the defrosted address.

5. **FAILURE** An attempt to distribute more than 1 billion tokens. An error should have occurred at some point. There were not errors, and tokens were distributed (Important.6)

6. **FAILURE** The test checking that totalSupply corresponds to the actual number of tokens issued before distribution (Important.4)

7. **FAILURE** The attempt to execute batchAsssignTokens several times. The test that all tokens have been distributed (Important.10)

Review of smart contract #2

Navi-token smart contract review #2

https://github.com/naviworld/navi-token

Important:

1. The addition is performed twice. Extra waste of gas.

   require(totalSupply.add(amount) <= MAX_NUM_NAVITOKENS);
   totalSupply = totalSupply.add(amount);

   can be replaced by

   totalSupply = totalSupply.add(amount);
   require(totalSupply <= MAX_NUM_NAVITOKENS);

2. mapIcedBalancesReserveAndTeamDefrosted[toAddress] = 0 in line 80, wastes the gas, since the default value is zero.

3. It assumes that defrosting will occur only after full distribution of tokens.
   We recommend you to add the batchAssignStopped == true check to canDefrostReserveAndTeam and in CanDefrostAdvars to exclude the possibility of distributing tokens after the tokens have already started to unfreeze in case of stopBatchAssign() has not been executed.

4. When you distribute tokens to the same address several times for ReserveAndTeam or Advisor, the recipient's address is added to the icedBalancesReserveAndTeam array every time and it turns out that there are several copies of this address in the array.

Code quality

1. Not correct indent of the line.

```
if (amountToRelease > 0) {
    mapIcedBalancesReserve
    mapIcedBalancesReserve
    balances[currentAddres

    Defrosted(currentAddre
}
```

2. The now> START_ICO_TIMESTAMP condition in canDefrostReserveAndTeam() and canDefrostAdvisors() is not required, since it always returns true after the first condition has been fulfilled (elapsedMonthsFromICOStart ()> = DEFROST_AFTER_MONTHS).

Testing

1. **SUCCESS** The check of defrosting the advisor tokens.

2. **SUCCESS** The attempt to perform the defrost function several times in one month and check that only the first defrost has worked.

3. **SUCCESS** The distribution of tokens, their defrosting for 30 months. Check that all tokens are defrosted correctly: at the specified time and in the right amount.

4. **SUCCESS** The defrost of the tokens for the ReserveAndTeam address, then their transfer from the address and re-defrost. The check that the number of defrosted tokens is independent of the current balance on the defrostable address.

5. **SUCCESS** Distribution of more than 1.000.000.000 tokens. The transaction failed.

6. **SUCCESS** The check that totalSupply corresponds to the actual number of tokens produced before the distribution.

7. **SUCCESS** Try to execute batchAsssignTokens several times. Verify that all tokens have been distributed successfully .

8. **FAILURE** Distribution of tokens after 7 months, when it is already possible to perform defrosting. The transaction should not have passed. https://gist.github.com/kolya-t/0ed85fd29582c08cd3d44922607c1845

9. **SUCCESS** The check that even after 40 months it defrosts even after 40 months the correct number of tokens. https://gist.github.com/kolya-t/abb62f26d8a74ea38852c24b381d0bcc

Review of smart contract #3

Navi-token smart contract review #3

https://github.com/naviworld/navi-token

Important:

1. The use of random data in tests is not recommended, as this test will not be repeated. We recommend using randomly assigned seed.

```
function randomValueAmount () {
  var low =  1;
  var high = 100000;
  return parseInt(Math.random() * (high - low) + low)
}
```

2. When you distribute tokens to the same address several times for ReserveAndTeam or Advisor, the recipient's address is added to the icedBalancesReserveAndTeam array every time and it turns out that there are several copies of this address in the array.

Code quality

1. TestNaviToken.js: Tests inside it() must be independent of each other. If one of the tests is removed or replaced, the result of the automatic tests should not be changed. It is recommended to add beforeEach or afterEach, that before each test the rollback is performed in time to the initial state.

Testing

1. **SUCCESS** The check of defrosting the advisor tokens.

2. **SUCCESS** The attempt to perform the defrost function several times in one month and check that only the first defrost has worked.

3. **SUCCESS** The distribution of tokens, their defrosting for 30 months. Check that all tokens are defrosted correctly: at the specified time and in the right amount.

4. **SUCCESS** The defrost of the tokens for the ReserveAndTeam address, then their transfer from the address and re-defrost. The check that the number of defrosted tokens is independent of the current balance on the defrostable address.

5. **SUCCESS** Distribution of more than 1.000.000.000 tokens. The transaction failed.

6. **SUCCESS** The check that totalSupply corresponds to the actual number of tokens produced before the distribution.

7. **SUCCESS** Try to execute batchAsssignTokens several times. Verify that all tokens have been distributed successfully.

8. **SUCCESS** Distribution of tokens after 7 months, when it is already possible to perform defrosting. The transaction should not have passed. https://gist.github.com/kolya-t/0ed85fd29582c08cd3d44922607c1845

9. **SUCCESS** The check that even after 40 months it defrosts even after 40 months the correct number of tokens. https://gist.github.com/kolya-t/abb62f26d8a74ea38852c24b381d0bcc

Results of contract audit

https://github.com/naviworld/navi-token

The information in this report is the list of recommendations what needs to be done to ensure the quality and security of the smart contract.

The OCEANICO experts conducted the verification of the smart contract. Based on the results, the customer's developers were given recommendations for optimizing the smart contract code.

OCEANICO experts conducted the audit of a smart contract in three iterations. Based on the results of each iteration, the customer's developers were given recommendations for eliminating errors committed while writing the code and possible vulnerabilities of the contract.

During three complete test run runs in the Ethereum test grid, all the errors found were eliminated by the developer. This smart contract corresponds to the specifications stated in the terms of reference and does not contain previously identified code and vulnerabilities errors.

On all issues regarding the audit and testing of a smart contract, we recommend contacting zoia@oceanico.io