SMART CONTRACT AUDIT REPORT

For

NEXXO Token (Order # 26OCT2018A)

Prepared By: Yogesh Padsala

Prepared on: 26/10/2018

audit@etherauthority.io

Prepared For: NEXXO

https://nexxo.io

Table of Content

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

2. Overview of the audit

The project has following file:

- NEXXO.sol

It contains approx **220** lines of Solidity code. All the functions and state variables are **not** well commented using the natspec documentation. However that does not raise any vulnerability, but it just increases the readability.

The audit was performed by Yogesh Padsala, from EtherAuthority Limited. Yogesh has extensive work experience of developing and auditing the smart contracts.

The audit was based on the solidity compiler 0.4.25+commit.59dbf8f1 with optimization enabled compiler in remix.ethereum.org

This audit was also performed verification of the details exist in whitepaper: https://nexxo.io/assets/staticassets/nexxo-whitepaper.pdf?v=2.0

Quick Stats:

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	Assert() misuse	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	“Out of Gas” Attack	Passed
Business Risk	Evil mint/burn	Passed
	The maximum limit for mintage not set	Passed
	“Fake Charge” Attack	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed
Auto Fuzzing		Passed

Overall Audit Result: PASSED

3. Attacks tested on the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

3.1: Over and under flows

This contract **does** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, and all the functions have strong validations, which prevented this attack.

3.2: Short address attack

Although this contract **is not vulnerable** to this attack, it is highly recommended to call functions after checking validity of the address from the outside client.

3.3: Visibility & Delegatecall

Delegatecall is not used in the contract thus it does not have this vulnerability. And visibility is also used properly.

3.4: Reentrancy / TheDAO hack

Use of “require” function and Checks-Effects-Interactions pattern in this smart contract mitigated this vulnerability.

3.5: Forcing ether to a contract

Here, the Smart Contract’s balance has never been used as guard, which mitigated this vulnerability

3.6: Denial Of Service (DOS)

There is no process consuming loops in the contracts which can be used for DoS attacks. Also, there is no progressing state based on external calls, and thus this contract is not prone to DoS.

4. Good things in the smart contract

4.1 Checks-Effects-Interactions pattern

While transferring tokens, this contract does all the process first and then transfers them. The same while doing other process too. This is very good practice which prevents malicious possibility. For example: transfer() and approve() functions.

4.2 Higher degree of admin control

This is always considered a great practice for the owner to halt or resume the token transfer and other process.

This is really useful in any unexpected event. So, admin can place safeguard on the contract until the issue is resolved. And he can resume it back again once the issue would be resolved.

4.3 Ability to mint new tokens

This contract enables admin/owner to generate new tokens manually. This is really helpful in any unexpected event such as tokens loss as well as for the crowdsale.

4.4 Functions input parameters passed

The functions in this contract verifies the validity of the input parameters, and this validations cannot be by-passed in anyway.

4.5 Custom error message in require()

This is great practice to put custom error messages in require() function, which makes debugging really easy.

4.6 Declare variables as constant

The value of variables at line number #50, #51, and #52, is not expected to change, thus it is good thing to declare them as constant, which helps reduce the gas cost.

5. Critical vulnerabilities found in the contract

Critical issues that could damage heavily the integrity of the contract. Some bug that would allow attackers to steal ether is a critical issue.

5.1: Mismatch of Information in whitepaper and code

The ICO details as of whitepaper are not implemented in the code. As like private sale, pre-sale, bonus, caps, etc.

On another hand, if that is intentional by having a separate smart contract for crowdsale, then this warning can be **safely ignored**. In this case, there should not be any necessity to have ICO data in this token contract code as they might be duplicate due to the data in separate crowdsale contract.

And if that is not the case, then ICO details must be implemented in this smart contract code.

6. Medium vulnerabilities found in the contract

Those vulnerabilities that could damage the contract but with some kind of limitations. Like a bug allowing people to modify a random variable.

=> No Medium vulnerabilities found

7. Low severity vulnerabilities found

Those do not damage the contract, but better to resolve and make code clean.

7.1: Unused variable

The variable ‘presaleStartDate’ at line number #64 is not used anywhere in the contract code.

If that is not intended to use it any longer, then it is good to remove it.

On another hand, if that variable serves any purpose, especially when called by other contracts as like crowdsale, then this warning can be ignored.

7.2: Fallback function

The fallback function does not do anything while ICO is running. In another words, it does not do anything apart from accepting Ether. Usually, that should send the tokens automatically to users and also should transfer the ether to owner wallet.

So again, if that is intentional, then this warning can be safely ignored. And if that is not the case, then it is recommended to implement such methods to enhance the user experience.

7.3: Token price in Fiat

Since EVM does not accept Fiat currency directly, specifying token price in USD or any other fiat currency will need to be converted into ETH.

And due to volatility of the ETH, there might be slight difference in the pricing.

On another hand, it is very good to specify the price of token in ETH, which is easier to implement in the code.

For example: 1 ETH = 1000 Tokens

8. Discussions and improvements

8.1 Prepare the code for the solidity version: 0.5.0

There are many improvements and upgrades will be introduced. As like:

Make your fallback functions external.

https://github.com/ethereum/solidity/blob/develop/Changelog.md#050-unreleased

8.3 approve() of ERC20 Standard

To prevent attack vectors like the one described here and discussed here, clients SHOULD make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. THOUGH the contract itself shouldn't enforce it, to allow backwards compatibility with contracts deployed before

8.4 While using SafeMath library

The SafeMath library is doing the great job to prevent overflow and underflow. However, it is recommended **NOT** to use it when overflow/underflow is impossible. Because please keep in mind that every unnecessary checks contribute to increased gas cost!

8.5 Consider using self-destruct function

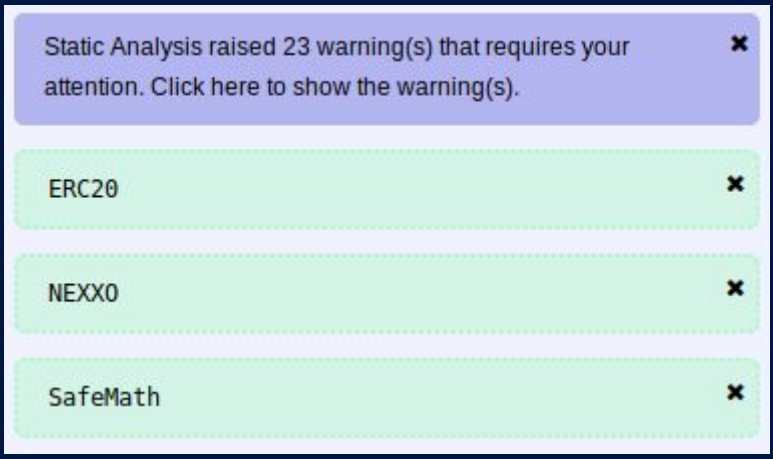
It many times happens, where contract owner would need to upgrade the contract or to add any important feature in the contract.

So, the only way that can be possible by creating brand new contract and destroying the old one. And that time, self-destruct comes to help.

9. Summary of the Audit

Overall the code performs good data validations as well as meets the correctness of data according to the information presented in the whitepaper: https://nexxo.io/assets/staticassets/nexxo-whitepaper.pdf?v=2.0

The compiler also displayed 23 warnings:



Now, we checked that the warnings in purple division, are due to their static analysis, which includes like gas estimations and all. So, it is important to supply correct gas values while calling various functions.

Those warnings can be safely ignored as should be taken care while calling the smart contract functions.

Please try to check the address and value of token externally before sending to the solidity code.

It is also encouraged to run bug bounty program and let community help to further polish the code to the perfection.