Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: NANXA Token
Platform: Binance Smart Chain
Language: Solidity
Date: August 25th, 2022

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the NANXA team to perform the Security audit of the NANXA Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on August 25th, 2022.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

NANXA coin is a standard BEP20 token smart contract. This audit only considers NANXA coin token smart contracts, and does not cover any other smart contracts on the platform.

Audit scope

Name	**Code Review and Security Analysis Report for NANXA Token Smart Contract**
Platform	**BSC / Solidity**
File	NANXA.sol
File MD5 Hash	AD9C8A0923042C6BAE553E740DF4EDC1
Audit Date	August 25th, 2022

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: NANXA • Symbol: NANXA • Decimals: 18 • Marketing Fee: 3% • Maximum Marketing Fee: 5% • totalsupply: 100 million • Swap Tokens at Amount: 20000 • Maximum Transaction Amount: 100 Million	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **“Secured”**. This token contract does contain owner control, which does not make it fully decentralized.

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 0 low and some very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical 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	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the NANXA Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the NANXA Token.

The NANXA Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

Documentation

We were given a NANXA Token smart contract code in the form of a BSCScan web link The hash of that code is mentioned above in the table.

As mentioned above, code parts are not well commented on. So it is not easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	name	read	Passed	No Issue
7	symbol	read	Passed	No Issue
8	decimals	read	Passed	No Issue
9	totalSupply	read	Passed	No Issue
10	balanceOf	read	Passed	No Issue
11	transfer	write	Passed	No Issue
12	allowance	read	Passed	No Issue
13	approve	write	Passed	No Issue
14	transferFrom	write	Passed	No Issue
15	increaseAllowance	write	Passed	No Issue
16	decreaseAllowance	write	Passed	No Issue
17	_transfer	internal	Passed	No Issue
18	_createTSupply	internal	Passed	No Issue
19	_approve	internal	Passed	No Issue
20	_beforeTokenTransfer	internal	Passed	No Issue
21	lockTheSwap	modifier	Passed	No Issue
22	_transfer	internal	Passed	No Issue
23	swapTokensForBNB	write	Passed	No Issue
24	excludeFromFees	write	Function input parameters lack of check	Refer Audit Findings
25	isExcludedFromFees	read	Passed	No Issue
26	setSwapTokensAtAmount	external	Critical operation lacks event log, Function input parameters lack of check	Refer Audit Findings
27	setSwapEnabled	write	access only Owner	No Issue
28	setMarketingWallet	external	Critical operation lacks event log, Function input parameters lack of check	Refer Audit Findings
29	setFee	write	Critical operation lacks event log	Refer Audit Findings
30	setMaxTransaction	write	Critical operation lacks event log,Function input parameters lack of check	Refer Audit Findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Critical operation lacks event log:

Missing event log for:

- setSwapTokensAtAmount()
- setMarketingWallet()
- setFee()
- setMaxTransaction()

Resolution: Please write an event log for listed events.

(2) Unlocked Compiler Version:

```
pragma solidity 0.8.16;
```

The contract uses the "^" prefix specifier, Use the Unlocked compiler version. Unlocked compiler version code of the smart contract, and that gives permission to the users to compile it one higher than a particular version.

Resolution: We suggest using that the compiler version is unlocked instead of the locked compiler version. The following line of code can be added to the project:

pragma solidity 0.8.16;

(3) SafeMath Library:

SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow/underflow.

Resolution: Remove the SafeMath library and use normal math operators, It will improve code size, and less gas consumption.

(4) Function input parameters lack of check:

Some functions require validation before execution.

Functions are:

- excludeFromFees()
- setSwapTokensAtAmount()
- setMarketingWallet()
- setMaxTransaction()

Resolution: We suggest using validation like for numerical variables that should be greater than 0 and for address type check variables that are not address(0).

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- excludeFromFees: Owner can exclude fees from address.
- setSwapTokensAtAmount: Owner can set swap token at amount.
- setSwapEnabled: Owner can set swap enabled status.
- setMarketingWallet: Owner can set marketing wallet address.
- setFee: Owner can set fee.
- setMaxTransaction: Owner can set maximum transaction amount.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a bscscan.com link and we have used all possible tests based on given objects as files. We have not observed any major issues in the token smart contract. So, **it's good to go for the mainnet deployment**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

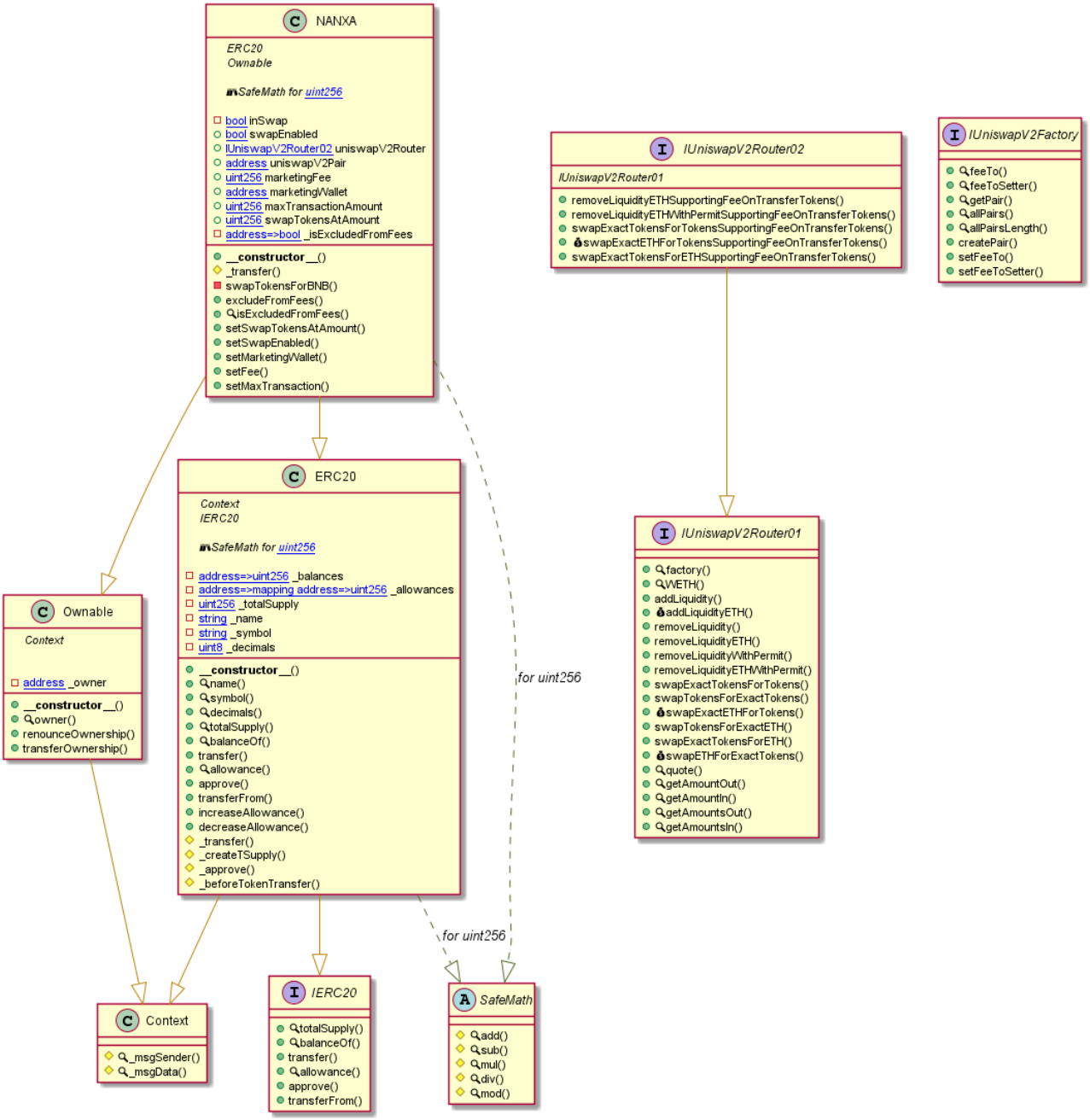
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - NANXA Token



Slither Results Log

Slither Log >> NANXA.sol

```
INFO:Detectors:
NANXA.setMarketingWallet(address)._newAddress (NANXA.sol#631) lacks a zero-check on :
        - marketingWallet = _newAddress (NANXA.sol#632)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in NANXA.constructor() (NANXA.sol#527-542):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (NAN
XA.sol#530-531)
        State variables written after the call(s):
        - _createTSupply(owner(),100000000 * 10 ** 18) (NANXA.sol#541)
                - _balances[account] = _balances[account].add(amount) (NANXA.sol#327)
        - excludeFromFees(owner(),true) (NANXA.sol#537)
                - _isExcludedFromFees[account] = excluded (NANXA.sol#613)
        - excludeFromFees(address(this),true) (NANXA.sol#538)
                - _isExcludedFromFees[account] = excluded (NANXA.sol#613)
        - excludeFromFees(marketingWallet,true) (NANXA.sol#539)
                - _isExcludedFromFees[account] = excluded (NANXA.sol#613)
        - _createTSupply(owner(),100000000 * 10 ** 18) (NANXA.sol#541)
                - _totalSupply = _totalSupply.add(amount) (NANXA.sol#326)
        - uniswapV2Pair = _uniswapV2Pair (NANXA.sol#534)
        - uniswapV2Router = _uniswapV2Router (NANXA.sol#533)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in NANXA._transfer(address,address,uint256) (NANXA.sol#544-589):
        External calls:
        - swapTokensForBNB(contractTokenBalance,marketingWallet) (NANXA.sol#572)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,_to,block.timestamp) (N
ANXA.sol#602-608)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (NANXA.sol#315)
                - super._transfer(from,to,amount) (NANXA.sol#587)
        - Transfer(sender,recipient,amount) (NANXA.sol#315)
                - super._transfer(from,address(this),fees) (NANXA.sol#580)
Reentrancy in NANXA.constructor() (NANXA.sol#527-542):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (NAN
```

```
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (NAN
XA.sol#530-531)
        Event emitted after the call(s):
        - ExcludeFromFees(account,excluded) (NANXA.sol#614)
                - excludeFromFees(address(this),true) (NANXA.sol#538)
        - ExcludeFromFees(account,excluded) (NANXA.sol#614)
                - excludeFromFees(marketingWallet,true) (NANXA.sol#539)
        - ExcludeFromFees(account,excluded) (NANXA.sol#614)
                - excludeFromFees(owner(),true) (NANXA.sol#537)
        - Transfer(address(0),account,amount) (NANXA.sol#328)
                - _createTSupply(owner(),100000000 * 10 ** 18) (NANXA.sol#541)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Context._msgData() (NANXA.sol#17-20) is never used and should be removed
SafeMath.mod(uint256,uint256) (NANXA.sol#152-154) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (NANXA.sol#160-163) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.4 (NANXA.sol#6) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function IUniswapV2Router01.WETH() (NANXA.sol#352) is not in mixedCase
Parameter NANXA.swapTokensForBNB(uint256,address)._to (NANXA.sol#591) is not in mixedCase
Parameter NANXA.setSwapEnabled(bool)._enabled (NANXA.sol#626) is not in mixedCase
Parameter NANXA.setMarketingWallet(address)._newAddress (NANXA.sol#631) is not in mixedCase
Parameter NANXA.setFee(uint256)._newFee (NANXA.sol#635) is not in mixedCase
Parameter NANXA.setMaxTransaction(uint256)._maxTxAmount (NANXA.sol#640) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (NANXA.sol#18)" inContext (NANXA.sol#12-21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```


Email: audit@EtherAuthority.io

```
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (NANXA
.sol#357) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).am
ountBDesired (NANXA.sol#358)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
NANXA.constructor() (NANXA.sol#527-542) uses literals with too many digits:
        - _createTSupply(owner(),100000000 * 10 ** 18) (NANXA.sol#541)
NANXA.slitherConstructorVariables() (NANXA.sol#501-646) uses literals with too many digits:
        - maxTransactionAmount = 100000000 * (10 ** 18) (NANXA.sol#512)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (NANXA.sol#60-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (NANXA.sol#69-73)
name() should be declared external:
        - ERC20.name() (NANXA.sol#210-212)
symbol() should be declared external:
        - ERC20.symbol() (NANXA.sol#218-220)
decimals() should be declared external:
        - ERC20.decimals() (NANXA.sol#225-227)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (NANXA.sol#251-254)
```

```
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (NANXA.sol#251-254)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (NANXA.sol#270-273)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (NANXA.sol#281-285)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (NANXA.sol#291-294)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (NANXA.sol#299-302)
isExcludedFromFees(address) should be declared external:
        - NANXA.isExcludedFromFees(address) (NANXA.sol#618-620)
setSwapEnabled(bool) should be declared external:
        - NANXA.setSwapEnabled(bool) (NANXA.sol#626-629)
setFee(uint256) should be declared external:
        - NANXA.setFee(uint256) (NANXA.sol#635-638)
setMaxTransaction(uint256) should be declared external:
        - NANXA.setMaxTransaction(uint256) (NANXA.sol#640-643)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:NANXA.sol analyzed (9 contracts with 75 detectors), 34 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

NANXA.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in NANXA.swapTokensForBNB(uint256,address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 610:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 626:12:

Gas & Economy

Gas costs:

Gas requirement of function NANXA.setSwapEnabled is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 645:4:

Gas costs:

Gas requirement of function NANXA.setMaxTransaction is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 659:4:

Miscellaneous

Constant/View/Pure functions:

IUniswapV2Router01.removeLiquidityETH(address,uint256,uint256,uint256,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 400:4:

Constant/View/Pure functions:

IUniswapV2Router01.removeLiquidityWithPermit(address,address,uint256,uint256,uint256,address,uint256,bool,uint8,
: Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 408:4:

Constant/View/Pure functions:

IUniswapV2Router01.swapExactTokensForETH(uint256,uint256,address[],address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 448:4:

Constant/View/Pure functions:

IUniswapV2Factory.setFeeTo(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 516:4:

Constant/View/Pure functions:

IUniswapV2Factory.setFeeToSetter(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 517:4:

Similar variable names:

ERC20._createTSupply(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 347:43:

No return:

IUniswapV2Factory.createPair(address,address): Defines a return type but never explicitly returns a value.
Pos: 514:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 655:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 661:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 161:20:

Solhint Linter

NANXA.sol

```
NANXA.sol:25:1: Error: Compiler version 0.8.16 does not satisfy the r
semver requirement
NANXA.sol:51:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
NANXA.sol:220:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
NANXA.sol:365:94: Error: Code contains empty blocks
NANXA.sol:371:5: Error: Function name must be in mixedCase
NANXA.sol:546:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
NANXA.sol:626:13: Error: Avoid to make time-based decisions in your
business logic
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io