Cyberscope

A TAC Security Company

Audit Report

NodeQ AI

July 2025

Network	ETH
Address	0x5EAD27FCC5Bc50aa100da635F8D1E6198ACEe71C
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	UAR	Unexcluded Address Restrictions	Unresolved
Minor / Informative	PART	Post-Liquidity Addition Remaining Tokens	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	LBE	Liquidity Balance Ensurance	Unresolved
Minor / Informative	MCM	Misleading Comment Messages	Unresolved
Minor / Informative	MVN	Misleading Variables Naming	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	NWES	Nonconformity with ERC-20 Standard	Unresolved
Minor / Informative	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
Minor / Informative	PPT	Potential Pre-Liquidity Trading	Unresolved
Minor / Informative	PTRP	Potential Transfer Revert Propagation	Unresolved
Minor / Informative	PVC	Price Volatility Concern	Unresolved
Minor / Informative	RLA	Redundant LP Approval	Unresolved

●	RRA	Redundant Repeated Approvals	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 4
Risk Classification 6
Review 7
Audit Updates 7
Source Files 7
Findings Breakdown 8
ST - Stops Transactions 9
Description 9
Recommendation 10
UAR - Unexcluded Address Restrictions 11
Description 11
Recommendation 12
PART - Post-Liquidity Addition Remaining Tokens 13
Description 13
Recommendation 13
CCR - Contract Centralization Risk 14
Description 14
Recommendation 14
IDI - Immutable Declaration Improvement 15
Description 15
Recommendation 15
LBE - Liquidity Balance Ensurance 16
Description 16
Recommendation 16
MCM - Misleading Comment Messages 17
Description 17
Recommendation 18
MVN - Misleading Variables Naming 19
Description 19
Recommendation 19
MEE - Missing Events Emission 20
Description 20
Recommendation 20
NWES - Nonconformity with ERC-20 Standard 21
Description 21
Recommendation 21
PLPI - Potential Liquidity Provision Inadequacy 22

Description 22
Recommendation 22
PPT - Potential Pre-Liquidity Trading 23
Description 23
Recommendation 23
PTRP - Potential Transfer Revert Propagation 24
Description 24
Recommendation 24
PVC - Price Volatility Concern 25
Description 25
Recommendation 25
RLA - Redundant LP Approval 26
Description 26
Recommendation 26
RRA - Redundant Repeated Approvals 27
Description 27
Recommendation 27
L02 - State Variables could be Declared Constant 28
Description 28
Recommendation 28
L04 - Conformance to Solidity Naming Conventions 29
Description 29
Recommendation 30
L16 - Validate Variable Setters 31
Description 31
Recommendation 31
Functions Analysis **32**
Inheritance Graph **34**
Flow Graph **35**
Summary **36**
Disclaimer **37**
About Cyberscope **38**

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	NodeQai
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://etherscan.io/address/0x5ead27fcc5bc50aa100da635f8d1e6198acee71c
Address	0x5ead27fcc5bc50aa100da635f8d1e6198acee71c
Network	ETH
Symbol	$NQAI
Decimals	9
Total Supply	10.000.000
Badge Eligibility	Must Fix Criticals

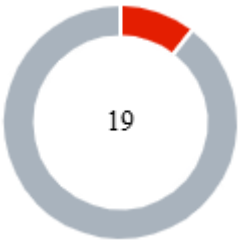
Audit Updates

Initial Audit	20 Jul 2025 https://github.com/cyberscope-io/audits/blob/main/nqai/v1/audit.pdf
Corrected Phase 2	29 Jul 2025

Source Files

Filename	SHA256
NodeQai.sol	f87db90e582260b4749a404ea32653074cd441bc4b89dc251bf46e8739529083

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	17	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	NODEQai.sol#L335,396,429
Status	Unresolved

Description

The contract owner has the authority to stop transactions, as detailed in the `PTRP` finding. As a result, the contract might operate as a honeypot.

Additionally, the contract does not allow the non-excluded addresses to transfer tokens. The restriction can be resumed once the contract owner enables them.

```
if (!tradingOpen) {
    require(
        _isExcludedFromFee[from] || _isExcludedFromFee[to],
        "Trading has not been enabled yet"
    );
}
...
if (ethInContract > 0) {
    _sendETHToFee(ethInContract);
}
...
function _sendETHToFee(uint256 amount) private {
    _tWallet.transfer(amount);
}
```

Recommendation

The team is advised to follow the recommendations outlined in the `PTRP` findings and implement the necessary steps to mitigate the identified risks, ensuring that the contract does not operate as a honeypot. Additionally, the owner should enable trading.

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership but it is non-reversible.

UAR - Unexcluded Address Restrictions

Criticality	Critical
Location	NODEQai.sol#L335,343
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

Specifically, the contract enforces restrictions on transfers from the liquidity pair when limits are in effect. However, since there is no mechanism to exclude specific addresses from these limitations, transferring tokens may encounter issues. The case is similar when trading is disabled, as it can prevent successful transfers.

Additionally, the `_isExcludedFromFee` mechanism does not actually exclude addresses from paying fees, which means some decentralized applications may experience difficulties during token transfers due to this limitation.

```
if (!tradingOpen) {
    require(_isExcludedFromFee[from] || _isExcludedFromFee[to],
"Trading has not been enabled yet");
}
if (from == uniswapV2Pair && to != address(uniswapV2Router) &&
!_isExcludedFromFee[to]) {
    if (limitEffect) {
        require(amount <= _maxTx, "Exceeds the maximum
transaction amount");
        require(balanceOf(to) + amount <= _maxWallet, "Exceeds
the maximum wallet size");
    }
    _buyTransactionCount++;
}
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments. Furthermore, the exclusion from restrictions should apply to all transactional limitations, not just specific ones like max transfer limitations, to avoid any unintended obstacles for excluded addresses.

PART - Post-Liquidity Addition Remaining Tokens

Criticality	Minor / Informative
Location	NodeQai.sol#L461
Status	Unresolved

Description

The contract uses the public function `addLiquidity` to add liquidity to the pair by sending tokens along with ETH. However, the amount of tokens sent is determined by the following calculation:

```
uint256 tokenAmount = balanceOf(address(this)) - (_tTotal * _initBT / 100);
```

This means that a portion of the tokens will not be sent to the pair and will remain in the contract. These remaining tokens will be held until a swap occurs, at which point the unsent tokens will be exchanged for ETH. The resulting ETH will then be transferred to the `_tWallet`.

Recommendation

It is recommended that the team makes the necessary changes so that when liquidity is added excess tokens do not remain in the contract.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	NODEQai.sol#L454,482,490,503,516,526
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function addLiquidity() external onlyOwner
function removeLimits() external onlyOwner returns (bool)
function setBT(uint256 newBT) external onlyOwner returns (bool)
function setST(uint256 newST) external onlyOwner returns (bool)
function openTrading() external onlyOwner returns (bool)
function clearStuckETH() external onlyOwner returns (bool)
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	NodeQai.sol#L199,215
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_tWallet
uniswapV2Pair
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

LBE - Liquidity Balance Ensurance

Criticality	Minor / Informative
Location	NODEQai.sol#L459
Status	Unresolved

Description

`addLiquidity` enables the owner to add liquidity using both ETH and the token. However, the contract's balance may not always contain sufficient ETH or tokens to facilitate the liquidity addition. Even when the contract holds both ETH and tokens, the available amounts may not meet the required ratio or be adequate for expanding the existing liquidity pool.

```
function addLiquidity() external onlyOwner {
    ...
    uint256 tokenAmount =  balanceOf(address(this)) - (_tTotal
* _initBT / 100);
    ...
    uniswapV2Router.addLiquidityETH{value:
address(this).balance}(
        address(this),
        tokenAmount,
        0,
        0,
        owner(),
        block.timestamp
    );
    ...
}
```

Recommendation

The team should implement checks to ensure that this function will be used only if the contract has enough tokens and ETH to create the intended ratio for liquidity.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	NodeQai.sol#L1,210,452
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

Specifically, the comment above `addLiquidity` mentions that the function creates a uniswap pair while it does not.

```
// Create Uniswap Pair (Once) & Add Liquidity
```

Also, in the constructor it is mentioned as a comment that `0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D` is the address in uniswap V2 router on sepolia and mainnet while this is true only for mainnet.

```
uniswapV2Router = IUniswapV2Router02(
    0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
); // This is Uniswap V2 router on Sepolia & Ethereum mainnet
```

Additionally the contract has been deployed with the following comment message.

```
/**
 *Submitted for verification at Etherscan.io on 2025-07-18
*/
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	NODEQai.sol#L335,343
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

Specifically, the mapping `_isExcludedFromFee` is not used to check if the user is excluded from fees in order to not take fees from their transaction.

```
if (!tradingOpen) {
    require(_isExcludedFromFee[from] || _isExcludedFromFee[to],
"Trading has not been enabled yet");
}
if (from == uniswapV2Pair && to != address(uniswapV2Router) &&
!_isExcludedFromFee[to]) {
    if (limitEffect) {
        require(amount <= _maxTx, "Exceeds the maximum
transaction amount");
        require(balanceOf(to) + amount <= _maxWallet, "Exceeds
the maximum wallet size");
    }
    _buyTransactionCount++;
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	NODEQai.sol#L428,435,454,482,490,503,516,526
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function _sendETHToFee(uint256 amount) private
function _swapTokensForEth(uint256 tokenAmount) private
lockTheSwap
function addLiquidity() external onlyOwner
function removeLimits() external onlyOwner returns (bool)
function setBT(uint256 newBT) external onlyOwner returns (bool)
function setST(uint256 newST) external onlyOwner returns (bool)
function openTrading() external onlyOwner returns (bool)
function clearStuckETH() external onlyOwner returns (bool)
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	NODEQai.sol#L329
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
require(amount > 0, "Transfer amount must be > 0");
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	NODEQai.sol#L442
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(tokenAmount,0,path,address(this),block.timestamp);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PPT - Potential Pre-Liquidity Trading

Criticality	Minor / Informative
Location	NODEQai.sol#L454,516
Status	Unresolved

Description

`addLiquidity` checks if `tradingOpen` is false in order to add liquidity. This ensures that this function can only be called before the owner enables trading. However, the contract does not ensure that liquidity has been added to the pair before the enabling of trading.

```
function addLiquidity() external onlyOwner {
    require(!tradingOpen, "Liquidity has already been
initialized");
    ...
}
...
function openTrading() external onlyOwner returns (bool) {
    require(!tradingOpen, "Trading has already been enabled");
    swapEnabled = true;
    tradingOpen = true;
    return true;
}
```

Recommendation

The contract should ensure that trading should only be enabled after liquidity has been added.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	NODEQai.sol#L396,429
Status	Unresolved

Description

The contract sends funds to a `_tWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
if (ethInContract > 0) {
    _sendETHToFee(ethInContract);
}
...
function _sendETHToFee(uint256 amount) private {
    _tWallet.transfer(amount);
}
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by sending the funds in a non-revertable way, while also ensuring that a gas exhaustion does not disrupt the execution flow.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	NODEQai.sol#L382
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `_tSwapThreshold` sets a threshold where the contract will trigger the swap functionality. The amount is limited up to `_maxTSwap` which is the maximum amount the contract can swap. If the variable stores a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
bool canSwap = !inSwap && (to == uniswapV2Pair) && swapEnabled
&& (contractTokenBalance > _tSwapThreshold) &&
(_buyTransactionCount > _manageSwapThreshold);
if (canSwap) {
    _swapTokensForEth(_minimum( amount,
_minimum(contractTokenBalance, _maxTSwap)));
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RLA - Redundant LP Approval

Criticality	Minor / Informative
Location	NODEQai.sol#L472
Status	Unresolved

Description

The contract approves uniswap router to use its LP tokens. However this approval is redundant since there is no functionality that uses the LP tokens. Furthermore, the LP tokens are transferred to the owner during the addition of liquidity meaning that the contract will not hold any LP tokens.

```
IERC20(uniswapV2Pair).approve(
    address(uniswapV2Router),
    type(uint).max
);
```

Recommendation

It is recommended to remove redundant code segments to reduce gas fees and optimize the code enhancing its readability.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	NODEQai.sol#L440
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
_approve(address(this), address(uniswapV2Router), tokenAmount);
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenA
mount,0,path,address(this),block.timestamp);
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	NodeQai.sol#L147,148,151,152,153,167,168,169,170
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
nt256 private _initBT              = 18;
nt256 private _initST              = 25;
nt256 private _reduceBTThreshold   = 30;
nt256 private _reduceSTThreshold   = 45;
nt256 private _manageSwapThreshold= 40;
nt256 public _maxTx            = 100000 * 10**_decimals;
nt256 public _maxWallet        = 100000 * 10**_decimals;
nt256 public _tSwapThreshold   = 10000 * 10**_decimals;
nt256 public _maxTSwap         = 100000 * 10**_decimals;
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	NodeQai.sol#L109,142,159,160,161,162,167,168,169,170
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
nction WETH() external pure returns (address);

dress private constant _bWallet = address(0xdead);
nt8 private constant _decimals     = 9;
nt256 private constant _tTotal     = 10000000 * 10**_decimals;
ring private constant _name        = unicode"NodeQ";
ring private constant _symbol      = unicode"$NQAI";
nt256 public _maxTx           = 100000 * 10**_decimals;
nt256 public _maxWallet       = 100000 * 10**_decimals;
nt256 public _tSwapThreshold  = 10000 * 10**_decimals;
nt256 public _maxTSwap        = 100000 * 10**_decimals;
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	NodeQai.sol#L68,199
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
_owner = msgSender;
_tWallet = tWalletAddress;
```

Recommendation

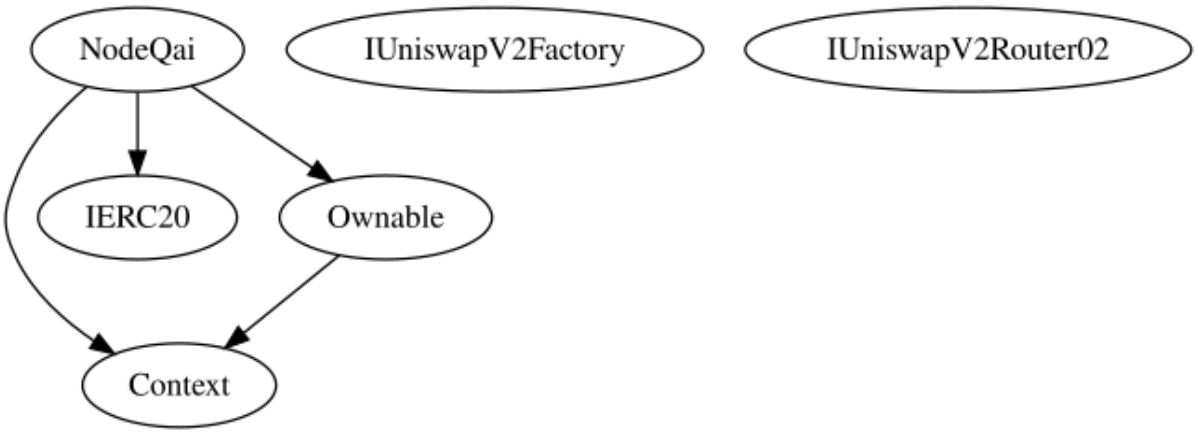
By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

Functions Analysis

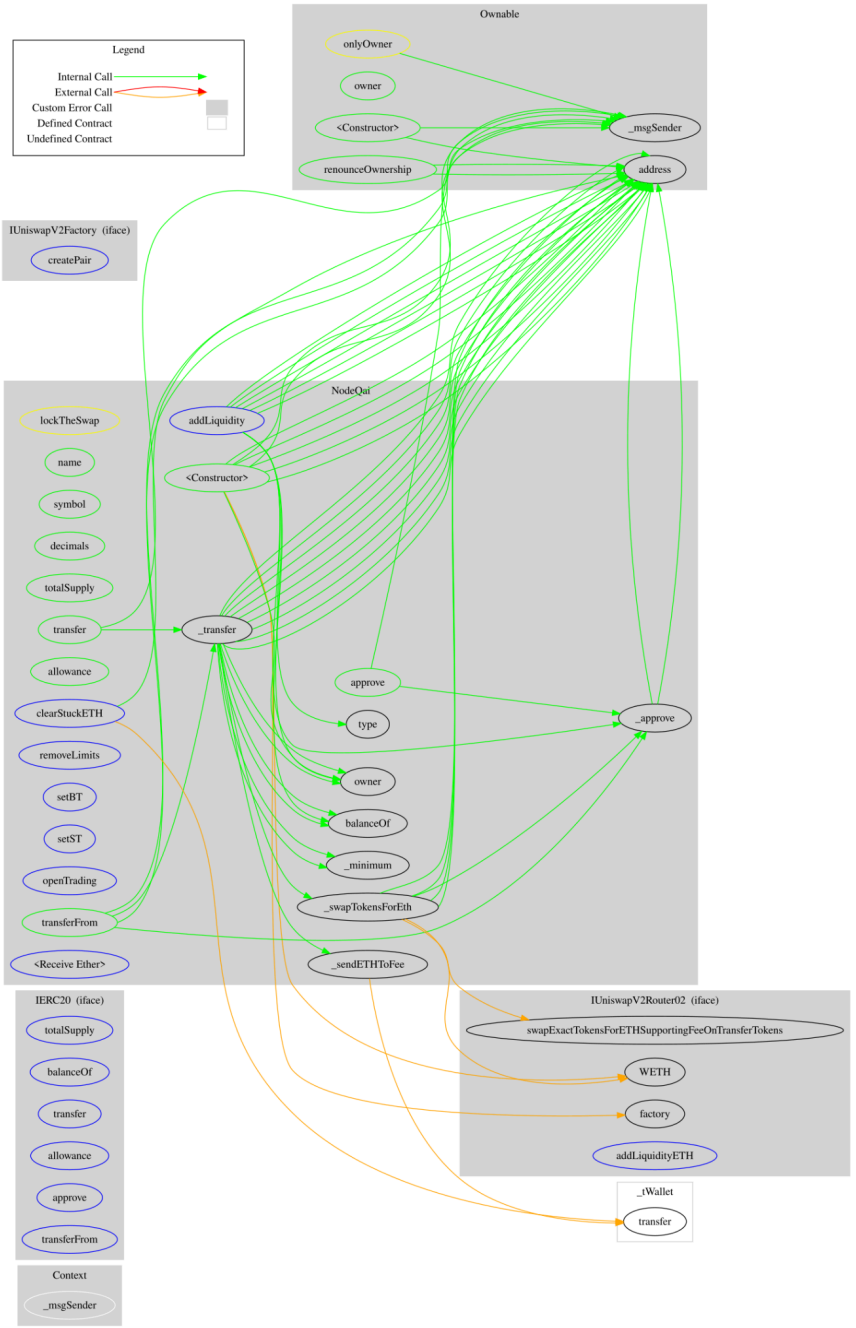
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
NodeQai	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	_minimum	Private		
	_sendETHToFee	Private	✓	
	_swapTokensForEth	Private	✓	lockTheSwap
	addLiquidity	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	setBT	External	✓	onlyOwner
	setST	External	✓	onlyOwner

	openTrading	External	✓	onlyOwner
	clearStuckETH	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
renounceOwnership
_msgSender
address
IUniswapV2Factory (iface)
createPair
NodeQai
lockTheSwap
name
symbol
decimals
totalSupply
transfer
allowance
clearStuckETH
removeLimits
setBT
setST
openTrading
transferFrom
<Receive Ether>
addLiquidity
<Constructor>
_transfer
approve
type
_approve
owner
balanceOf
_minimum
_swapTokensForEth
_sendETHToFee
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
addLiquidityETH
_tWallet
transfer

Summary

NodeQ AI contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. There is also a limit of max 25% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a TAC blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A **TAC Security** Company

The Cyberscope team

cyberscope.io