Cyberscope

A TAC Security Company

Audit Report

NOBLEBLOCKS

July 2025

Network	ETH
Address	0xe3e72Ae23dd5bb0EFd7dfE733A58bF6aD57C3945
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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	IFT	Inconsistent Fee Threshold	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	ZBS	Zero Balance Swap	Unresolved
●	PSU	Potential Subtraction Underflow	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MU	Modifiers Usage	Unresolved
●	PAMAR	Pair Address Max Amount Restriction	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L08	Tautology or Contradiction	Unresolved

●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

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	NOBLEBLOCKS
Compiler Version	v0.8.30+commit.73712a01
Optimization	200 runs
Explorer	https://etherscan.io/address/0xe3e72ae23dd5bb0efd7dfe733a58bf6ad57c3945
Address	0xe3e72ae23dd5bb0efd7dfe733a58bf6ad57c3945
Network	ETH
Symbol	NOBL
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Must Fix Criticals

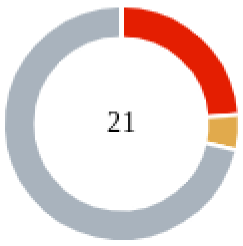
Audit Updates

Initial Audit	14 Jul 2025

Source Files

Filename	SHA256
NOBLEBLOCKS.sol	884bcb13590606ffaeab60518603f4599e9c5c64111f7a018c5368bd38b9f8c8

Findings Breakdown



●	Critical	5
●	Medium	1
●	Minor / Informative	15

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	5	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	NOBLEBLOCKS.sol#L946
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users. The owner may take advantage of it by invoking the `setPaused` method. As a result, the contract may operate as a honeypot.

In addition transactions may stop as described in the findings `ZBS` , `PAMAR` , `IFT` and `PTRP` .

```
function setPaused(bool _paused) external onlyOwner {
paused = _paused;
emit Paused(_paused);
}
```

Recommendation

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, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	NOBLEBLOCKS.sol#L1159
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setFee` function with a high percentage value. Specifically the owner may set the fees up to 90%.

```
require (0 < _newadminFee + _newfundFee && _newadminFee + _newfundFee <= 9000,
"need to be strictly positive and equal or smaller than 90%");
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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, which will eliminate the threats but it is non-reversible.

IFT - Inconsistent Fee Threshold

Criticality	Critical
Location	NOBLEBLOCKS.sol#L1015
Status	Unresolved

Description

The contract implements a fee mechanism on transfers. The fees to be withheld are tracked via the `totalFee` variable, which is defined as the sum of `liquidityFee` , `adminFee` , and `fundFee` . The contract includes a check to ensure that `totalFee` does not exceed a value of `5000` . Nevertheless, the `setFee` method allows this limit to be surpassed. As a result, if the defined fee values exceed the allowed threshold, all fee-inducing transfers will fail.

```
require (0 < _newadminFee + _newfundFee && _newadminFee + _newfundFee <= 9000,
"need to be strictly positive and equal or smaller than 90%");
```

```
require(totalFee <= 5000, "Total fee exceeds 50%");
```

Recommendation

The team is advised to improve consistency by ensuring proper validation of fee values across all relevant methods. Furthermore, eliminating redundant checks such as those found within the transfer method would improve gas efficiency and overall contract optimization.

PMRM - Potential Mocked Router Manipulation

Criticality	Critical
Location	NOBLEBLOCKS.sol#L954
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
function setUniswapV2Router(address _uniswapV2Router) external onlyOwner
{
...
}
```

Recommendation

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, which will eliminate the threats but it is non-reversible.

ZBS - Zero Balance Swap

Criticality	Critical
Location	NOBLEBLOCKS.sol#L1069
Status	Unresolved

Description

The contract implements the `sendDividends` function as part of its transaction flow. This function transfers native tokens received from fee-to-token swaps. It requires that the contract's balance of native tokens be non-zero. However, it is possible for `sendDividends` to be invoked even when the contract holds no native tokens. Specifically, if the pool holds no liquidity or the swap returns no tokens the function will be invoked with a zero amount. In addition, if the `liquidityFee` is set to zero or the `swapTokensAtAmount` is also zero, it is possible that the `swapAndLiquify` method does not process the swap before the `sendDividends` method is called. This will result in all transactions invoking the method to revert.

```
function sendDividends() private nonReentrant {
uint256 ethBalance = address(this).balance;
require(ethBalance > 0, "No ETH to distribute");
...
}
```

Recommendation

It is advisable to ensure that the contract maintains a sufficient native token balance before invoking the function. This can be achieved by conditionally checking the contract balance at the start of the execution.

PSU - Potential Subtraction Underflow

Criticality	Medium
Location	NOBLEBLOCKS.sol#L1018
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

In particular, the contract subtracts the calculated fees from the transferred amount. The fees are calculated as a percentage of the amount using the formula:

```
fee = amount * (liquidityFee + adminFee + fundFee) / 10000;
```

However, it is possible for the ratio `(liquidityFee + adminFee + fundFee) / 10000` to exceed `1`. In such cases, the fee to be withheld exceeds the transferred amount. If this occurs, the subtraction will underflow, causing the transaction to revert.

```
function setFee (uint16 _newLPFee, uint16 _newadminFee, uint16 _newfundFee)
external onlyOwner {
require(0 <= _newLPFee && _newLPFee <= 9000, "setFee: Fee should be equal or
smaller than 90%");
require(0 <= _newadminFee && _newadminFee <= 9000, "setFee: Fee should be
equal or smaller than 90%");
require(0 <= _newfundFee && _newfundFee <= 9000, "setFee: Fee should be equal
or smaller than 90%");
require (0 < _newadminFee + _newfundFee && _newadminFee + _newfundFee <= 9000,
"need to be strictly positive and equal or smaller than 90%");
liquidityFee = _newLPFee;
adminFee = _newadminFee;
fundFee = _newfundFee;

emit SetFee(liquidityFee, adminFee, fundFee);
}
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L946,954,1090,1098,1109,1119,1131,1139,1147,1155,1169,1179,1190
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 setPaused(bool _paused) external onlyOwner {...}
function setUniswapV2Router(address _uniswapV2Router) external onlyOwner {...}
function setSwapAtAmount (uint256 amount) external onlyOwner {...}
function setAdminWallet (address _address) external onlyOwner {...}
function setFundWallet (address _address) external onlyOwner {...}
function changeOwner (address _address) external onlyOwner {...}
function setExcludeWallet (address _address, bool _value) external
onlyOwner{...}
function setExcludeLimitWallet (address _address, bool _value) external
onlyOwner{...}
function setLimit (uint256 _limit) external onlyOwner {...}
function setFee (uint16 _newLPFee, uint16 _newadminFee, uint16 _newfundFee)
external onlyOwner {...}
function recoverTokens(address tokenAddress, uint256 amount) external
onlyOwner {...}
function recoverETH(address to, uint256 amount) external onlyOwner {...}
function setOnlySellFee(bool _value) external onlyOwner {...}
```

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	NOBLEBLOCKS.sol#L920
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.

```
multisigWallet
```

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.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L915,916,955,964,969,1099,1110,1120,1132,1140,1170,1181
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
_address != address(0)
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L1023,1139
Status	Unresolved

Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
if (!isLimitExcluded[to]){
require((balanceOf(to) + amount) <= balanceLimit, "maxlimit");
}
```

```
function setExcludeLimitWallet (address _address, bool _value) external
onlyOwner{
require(_address != address(0), "setExcludeLimitWallet: Invalid address");
require(isLimitExcluded[_address] != _value, "Same limit exclusion value
provided");
isLimitExcluded[_address] = _value;
emit ExcludeLimitWallet(_address, _value);
}
```

Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L1043,1057
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.

```
function swapTokensForEth(uint256 tokenAmount) private {
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = uniswapV2Router.WETH();
_approve(address(this), address(uniswapV2Router), tokenAmount);
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
tokenAmount,
0,
path,
address(this),
block.timestamp + 200
);
}
```

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private
{
_approve(address(this), address(uniswapV2Router), tokenAmount);
uniswapV2Router.addLiquidityETH{value: ethAmount}(
address(this),
tokenAmount,
0,
0,
multisigWallet,
block.timestamp + 200
);
}
```

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L1077,1078
Status	Unresolved

Description

The contract sends funds to an `adminWallet` and a `fundWallet` as part of the transfer flow. These addresses can either be a wallet address or a contract. If these addresses belong 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. In addition, both addresses may implement logic that consumes all gas of the incoming transactions. This will lead all tranfers to fail.

```
(bool successAdmin, ) = adminWallet.call{value: adminETH}("");
(bool successFund, ) = fundWallet.call{value: fundETH}("");
```

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 not allowing set contract addresses or by sending the funds in a non-revertable way. In addition, the contract should ensure it forwards a fixed amount of gas during the execution of the external call.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L1000
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. However the contract swaps its entire balance once that threshold is exceeded. It is therefore possible that the contract swaps a large 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.

```
swapAndLiquify(contractTokenBalance);
```

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.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L1047,1058
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);
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L683,684,701,737,877,946,954,976,1098,1109,1119,1131,1139,1147,1155,1190
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.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);


...

uint256 public immutable TOTAL_SUPPLY = 1_000_000_000 *
10**decimals();
function setPaused(bool _paused)
function setUniswapV2Router(address _uniswapV2Router)
function burnToken (uint256 _amount)
...
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L1156,1157,1158
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(0 <= _newLPFee && _newLPFee <= 9000, "setFee: Fee should be equal or
smaller than 90%");
require(0 <= _newadminFee && _newadminFee <= 9000, "setFee: Fee should be
equal or smaller than 90%");
require(0 <= _newfundFee && _newfundFee <= 9000, "setFee: Fee should be equal
or smaller than 90%");
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L92,117
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _reentrancyGuardEntered() internal view returns (bool) {
        return _status == ENTERED;
    }

function _contextSuffixLength() internal view virtual returns (uint256)
{
        return 0;
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L984,1009
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(paused == false, "Contract is Paused");
OnlySellFee == false || (isOnlySellFee == true && to ==
uniswapV2Pair))
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L1033,1040
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 half = tokens / 2;
addLiquidity(otherHalf, (newBalance*half)/halfPlusDividends);
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L10
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.20;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	NOBLEBLOCKS.sol#L1173
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
ERC20(tokenAddress).transfer(_msgSender(), amount);
```

Recommendation

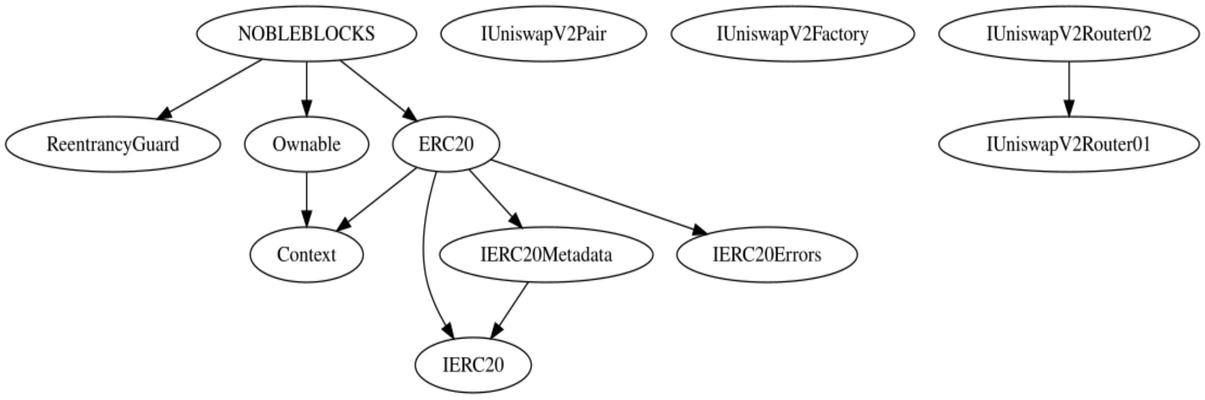
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

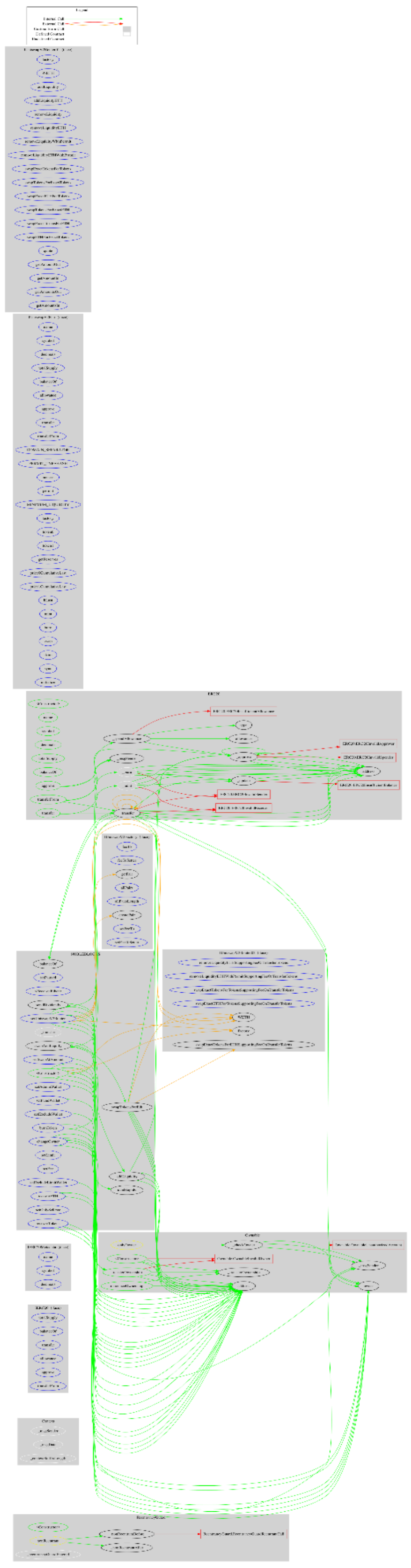
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
NOBLEBLOCKS	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20 Ownable
	setPaused	External	✓	onlyOwner
		External	Payable	-
	setUniswapV2Router	External	✓	onlyOwner
	burnToken	External	✓	onlyOwner
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	nonReentrant
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	sendDividends	Private	✓	nonReentrant
	setSwapAtAmount	External	✓	onlyOwner
	setAdminWallet	External	✓	onlyOwner
	setFundWallet	External	✓	onlyOwner
	changeOwner	External	✓	onlyOwner
	setExcludeWallet	External	✓	onlyOwner

	setExcludeLimitWallet	External	✓	onlyOwner
	setLimit	External	✓	onlyOwner
	setFee	External	✓	onlyOwner
	recoverTokens	External	✓	onlyOwner
	recoverETH	External	✓	onlyOwner
	setOnlySellFee	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

NOBLEBLOCKS 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 stopping transactions and manipulating the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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 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