Cyberscope

Audit Report

NOBLEBLOCKS

March 2024

Repository https://github.com/NOBLBLOCKS/NOBLEBLOCKSToken

Commit b1a2b6587eb66c28b4bb1d0c86c058c733839c22

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
	MEE	Missing Events Emission	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PMRM	Potential Mocked Router Manipulation	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Review

Contract Name	NOBLEBLOCKS
Repository	https://github.com/NOBLBLOCKS/NOBLEBLOCKSToken
Commit	b1a2b6587eb66c28b4bb1d0c86c058c733839c22
Testing Deploy	https://testnet.bscscan.com/address/0xc300dca6bc270752e73a694673b8974feef11c64
Symbol	NOBL
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	09 Mar 2024 https://github.com/cyberscope-io/audits/blob/main/nobl/v1/audit.pdf
Corrected Phase 2	22 Mar 2024

Source Files

Filename	SHA256
contracts/NOBLEBLOCKSToken.sol	5a2f2f2fae0dcc666df8d6eb5d64b3242c8202a356aae08e6fd3e2c7a5008753

Findings Breakdown

Critical	1
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/NOBLEBLOCKS.sol#L906
Status	Unresolved

Description

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

Recommendation

It is advised to implement checks that limit the contract's ability to unilaterally stop transactions, as outlined in `PTRP` section. This precaution helps mitigate the risk of the contract being used as a honeypot, ensuring a more secure and trustworthy environment for users.

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L1120
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 `setFees` function with a high percentage value.

```
    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.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp + 200
        );
    }
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKS.sol#L1087,1098
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 setExcludeWallet(
        address _address,
        bool _value
    ) external onlyOwner {
        require(
            isExcluded[_address] != _value,
            "Same exclusion value provided"
        );
        isExcluded[_address] = _value;
    }

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

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L1003
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.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
            tokenAmount,
            0,
            path,
            address(this),
            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.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L906
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 {
        IUniswapV2Router02 _UniswapV2Router =
IUniswapV2Router02(
            _uniswapV2Router
        );
        IUniswapV2Factory factory = IUniswapV2Factory(
            _UniswapV2Router.factory()
        );

        // Check if pair already exists
        address existingPair = factory.getPair(
            address(this),
            _UniswapV2Router.WETH()
        );

        address _UniswapV2Pair;
        if (existingPair == address(0)) {
            // If pair does not exist, create it
            _UniswapV2Pair = factory.createPair(
                address(this),
                _UniswapV2Router.WETH()
            );
        } else {
            // If pair exists, use existing pair address
            _UniswapV2Pair = existingPair;
        }
        require(_UniswapV2Pair != address(0), "Pair address set
to zero");
        uniswapV2Router = _UniswapV2Router;
        uniswapV2Pair = _UniswapV2Pair;
        isLimitExcluded[_UniswapV2Pair] = true;
    }
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L901,905
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function excludeFromFees(address account, bool excluded)
external onlyOwner {
        isExcluded[account] = excluded;
    }

    function excludeFromLimit(address account, bool excluded)
external onlyOwner {
        isLimitExcluded[account] = excluded;
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKS.sol#L559,561,592,663,861,906,937,1056,1066,1077,1088,1089,1099,1100,1109,1121,1122,1123
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);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
uint8 private constant percentageOfMaximumTokensToAccumate = 10
address _uniswapV2Router
uint256 _amount

function _burnToken(uint256 _amount) external onlyOwner {
        _burn(owner(), _amount);
    }
address _address
bool _value
uint256 _limit
uint16 _newLPFee

...
```

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/v0.8.17/style-guide.html#naming-convention.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKS.sol#L993,1000
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) /
halfPlusDividents)
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ReentrancyGuard	Implementation			
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-

	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	

	_beforeTokenTransfer	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-

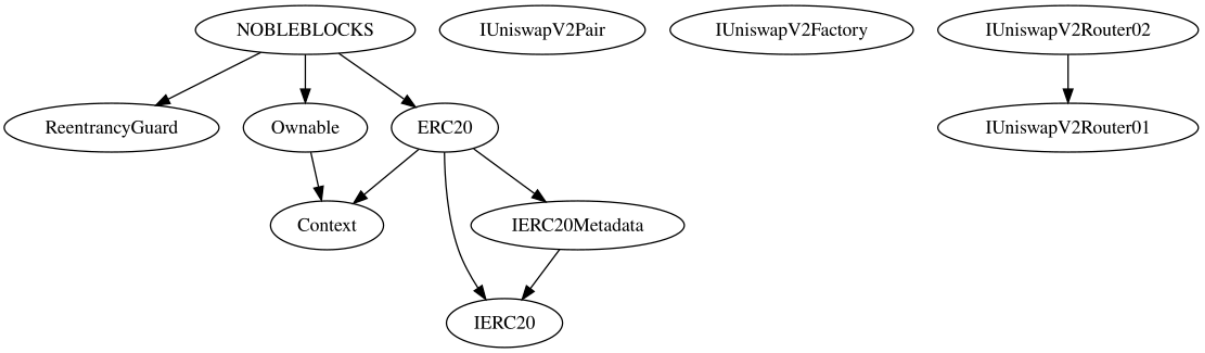
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-

	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-

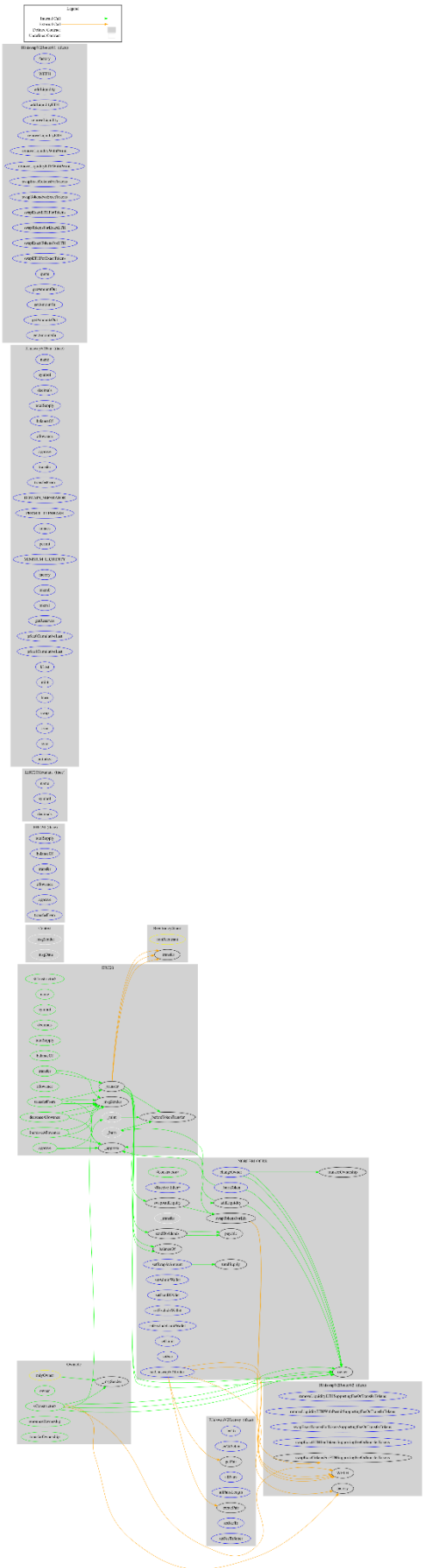
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
NOBLEBLOCKS	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20
		External	Payable	-
	setUniswapV2Router	External	✓	onlyOwner
	_burnToken	External	✓	onlyOwner
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	

	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

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 stop transactions and manipulate 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. There is also a limit of max 30% 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 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.

The Cyberscope team

https://www.cyberscope.io