Cyberscope

Audit Report

NOBLEBLOCKS

March 2024

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

Commit c8ac75c6dbb4e8a9632cd4d508b9641996ad65e1

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	IFM	Inefficient Fee Management	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RCF	Redundant Code Functionality	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Repository	https://github.com/NOBLBLOCKS/NOBLEBLOCKSToken
Commit	c8ac75c6dbb4e8a9632cd4d508b9641996ad65e1
Testing Deploy	https://goerli.etherscan.io/address/0x53a17a97cf2a5298d5725f69f7f4ccc4d4dc2fa1
Badge Eligibility	Yes

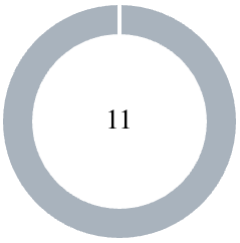
Audit Updates

Initial Audit	09 Mar 2024

Source Files

Filename	SHA256
contracts/NOBLEBLOCKSToken.sol	2940a9e1b469b11b26866bc2da6d73de6021c0dea2470e4d35abd1f36e564e19

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L893
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 setFees(uint16 _liquidityFee, uint16 _adminFee,
uint16 _fundFee) external onlyOwner {
        require(_liquidityFee + _adminFee + _fundFee <=
TOTAL_FEE_LIMIT, "Total fee exceeds max limit");

        liquidityFee = _liquidityFee;
        adminFee = _adminFee;
        fundFee = _fundFee;
    }
```

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.

IFM - Inefficient Fee Management

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L882
Status	Unresolved

Description

The contract is designed to handle transactions with associated fees, specifically declaring three different fee variables the `liquidityFee`, `adminFee`, and `fundFee`. However, the implementation aggregates these fees into a single total fee without utilizing the distinct functionalities or purposes originally implied by the separate fee variables. This approach simplifies the fee calculation and collection process but overlooks the potential for differentiated fee handling, which could benefit various aspects of the contract's ecosystem. By consolidating the fees into a single sum, the contract effectively nullifies the need for multiple fee variables, as the distinction between them does not influence the transaction process or fee allocation. Consequently, the different fee structure is not necessary, since the contract operates as if it is required only one fee variable.

```
   function _transfer(address from, address to, uint256 amount)
internal override {
        ...
        uint256 fee = 0;

        if(!isExcluded[from] && !isExcluded[to]){
            fee = amount * (liquidityFee + adminFee + fundFee) /
10000;
            transferAmount = amount - fee;
        }

        if(fee > 0) {
            super._transfer(from, fundWallet, fee);
        }

        super._transfer(from, to, transferAmount);
    }
```

Recommendation

It is recommended to either refactor the contract to utilize the distinct fee variables in a manner that reflects their intended purposes or simplify the fee structure by consolidating the separate fees into a single variable if differentiation is not necessary. If the original intent was to support different functionalities or allocations for each fee type, the contract should include specific logic to separately handle, calculate, and distribute each fee according to its unique role. This could involve directing each fee to different wallets or funds, applying them under different conditions, or using them to interact with other contracts or ecosystems. If, however, the differentiation of fees does not serve a strategic purpose, consolidating them into a single variable can streamline the contract's code, reduce complexity, and potentially lower gas costs for transactions.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L887
Status	Unresolved

Description

The contract sends funds to a `fundWallet` 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.

```
super._transfer(from, fundWallet, fee);
```

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.

RCF - Redundant Code Functionality

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L853,909,905
Status	Unresolved

Description

The contract declares variables and functions that are not utilized within its codebase, leading to redundant code. Specifically, the contract declares a `developmentWallet` and provides a function `setDevelopmentWallet` to update its address. Despite these declarations, the `developmentWallet` is not referenced or used in any operational part of the contract's logic. This implies that the variable, along with its associated setter function, serves no practical purpose within the current contract's implementation. Additionally, the contract implements the `excludeFromLimit` function which updates a mapping `isLimitExcluded` to track addresses that are excluded from certain unspecified limits. However, there is no subsequent functionality that leverages the `isLimitExcluded` mapping to alter behavior based on an account's exclusion status. This indicates a disconnection between the intended design elements and their actual application within the contract.

```
   developmentWallet =
0xeE9B6aa7196C1d9a3Bb0dE858E1E0aB81D0cd0e0;
    ...
    function setDevelopmentWallet(address _developmentWallet)
external onlyOwner {
        developmentWallet = _developmentWallet;
    }

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

Recommendation

It is recommended to review and refactor the contract's codebase to remove or appropriately integrate the unused `developmentWallet` and the `isLimitExcluded`

mapping functionalities. For the `developmentWallet`, if the intention was to utilize it for development-related expenses or operations, it's essential to implement the corresponding logic that leverages this wallet. Otherwise, removing the variable and its setter function can help streamline the contract and reduce unnecessary complexity. Regarding the `isLimitExcluded` functionality, it is crucial to either implement the logic that checks this mapping and alters contract behavior based on an account's exclusion status or remove the mapping and its associated function if they are unnecessary.

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/NOBLEBLOCKSToken.sol#L275,278,373,387,391,739,762,766,893,909,913
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 __Context_init() internal onlyInitializing {
    }

function __Context_init_unchained() internal onlyInitializing {
    }

bytes32 private constant ERC20StorageLocation =
0x52c63247e1f47db19d5ce0460030c497f067ca4cebf71ba98eeadabe20bac
e00;


function __ERC20_init(string memory name_, string memory
symbol_) internal onlyInitializing {
        __ERC20_init_unchained(name_, symbol_);
    }
...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L896,901,905
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
   function setFees(uint16 _liquidityFee, uint16 _adminFee, uint16
_fundFee) external onlyOwner {
        require(_liquidityFee + _adminFee + _fundFee <=
TOTAL_FEE_LIMIT, "Total fee exceeds max limit");

        liquidityFee = _liquidityFee;
        adminFee = _adminFee;
        fundFee = _fundFee;
    }

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

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

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L236,252,275,278,288,589
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 _disableInitializers() internal virtual {
        // solhint-disable-next-line var-name-mixedcase
        InitializableStorage storage $ =
_getInitializableStorage();

        if ($._initializing) {
            revert InvalidInitialization();
        }
        if ($._initialized != type(uint64).max) {
            $._initialized = type(uint64).max;
            emit Initialized(type(uint64).max);
        }
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L910,914
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.

```
developmentWallet = _developmentWallet
fundWallet = _fundWallet;
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L268,376,742
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            $.slot := INITIALIZABLE_STORAGE
        }

assembly {
            $.slot := ERC20StorageLocation
        }


assembly {
            $.slot := OwnableStorageLocation
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/NOBLEBLOCKSToken.sol#L8
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.

Functions Analysis

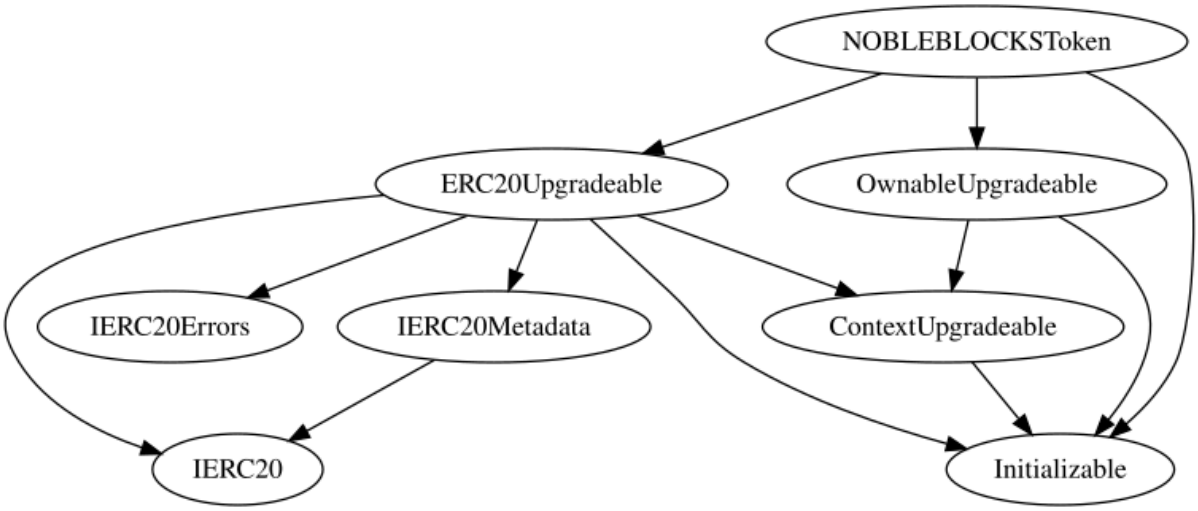
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Initializable	Implementation			
	_checkInitializing	Internal		
	_disableInitializers	Internal	✓	
	_getInitializedVersion	Internal		
	_isInitializing	Internal		
	_getInitializableStorage	Private		

ContextUpgradeable	Implementation	Initializable		
	__Context_init	Internal	✓	onlyInitializing
	__Context_init_unchained	Internal	✓	onlyInitializing
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
IERC20Errors	Interface			
ERC20Upgradeable	Implementation	Initializable, ContextUpgradeable, IERC20, IERC20Metadata, IERC20Errors		
	_getERC20Storage	Private		
	__ERC20_init	Internal	✓	onlyInitializing
	__ERC20_init_unchained	Internal	✓	onlyInitializing
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-

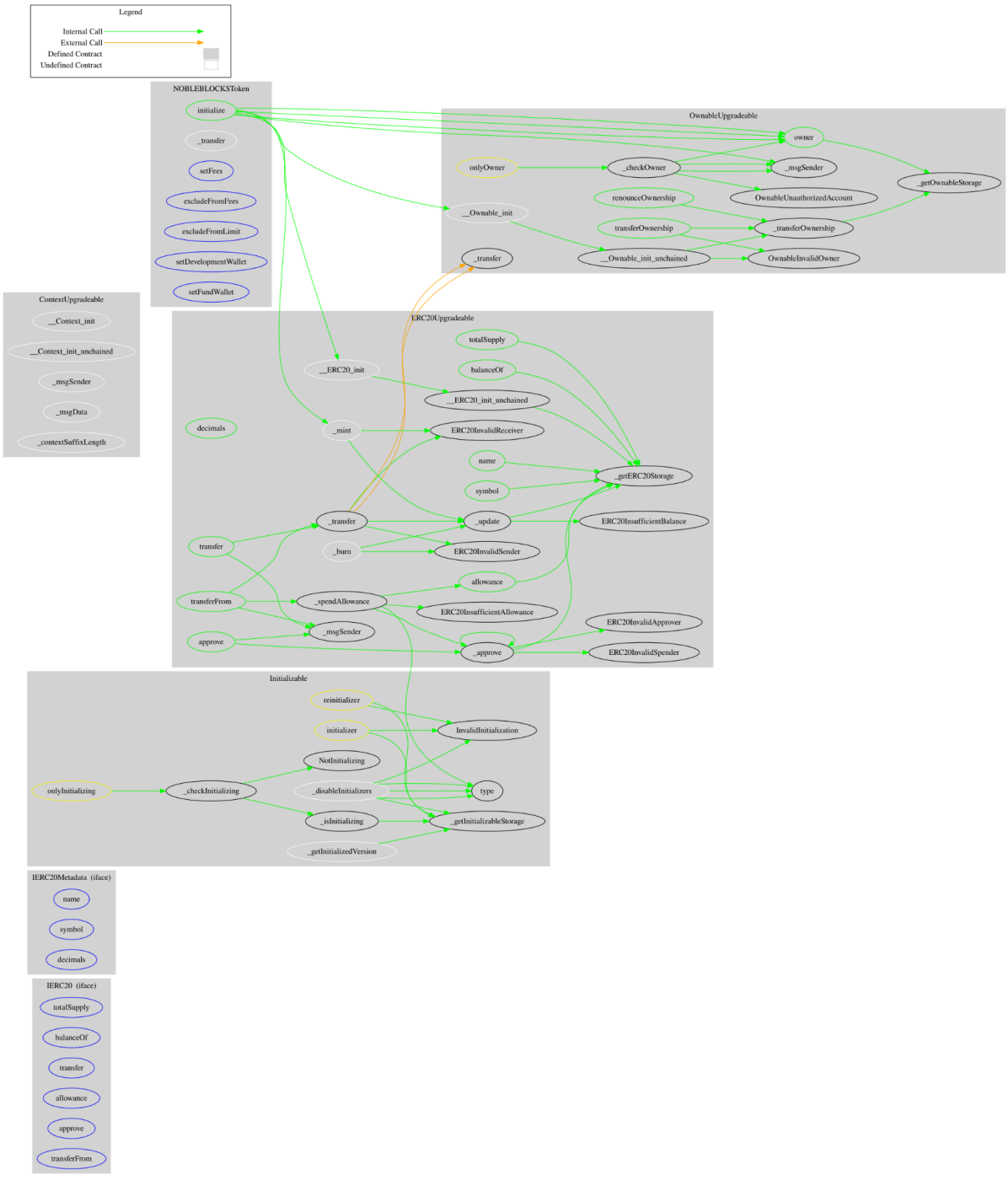
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
OwnableUpgradeable	Implementation	Initializable, ContextUpgradeable		
	_getOwnableStorage	Private		
	__Ownable_init	Internal	✓	onlyInitializing
	__Ownable_init_unchained	Internal	✓	onlyInitializing
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
NOBLEBLOCKSToken	Implementation	Initializable, ERC20Upgradeable, OwnableUpgradeable		

	initialize	Public	✓	initializer
	_transfer	Internal	✓	
	setFees	External	✓	onlyOwner
	excludeFromFees	External	✓	onlyOwner
	excludeFromLimit	External	✓	onlyOwner
	setDevelopmentWallet	External	✓	onlyOwner
	setFundWallet	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
NOBLEBLOCKSToken
initialize
_transfer
setFees
excludeFromFees
excludeFromLimit
setDevelopmentWallet
setFundWallet
OwnableUpgradeable
owner
onlyOwner
_checkOwner
_msgSender
_getOwnableStorage
renounceOwnership
OwnableUnauthorizedAccount
__Ownable_init
transferOwnership
_transferOwnership
_transfer
__Ownable_init_unchained
OwnableInvalidOwner
ContextUpgradeable
__Context_init
__Context_init_unchained
_msgSender
_msgData
_contextSuffixLength
ERC20Upgradeable
totalSupply
__ERC20_init
balanceOf
__ERC20_init_unchained
decimals
_mint
ERC20InvalidReceiver
name
_getERC20Storage
symbol
_transfer
_update
ERC20InsufficientBalance
transfer
_burn
ERC20InvalidSender
allowance
transferFrom
_spendAllowance
ERC20InsufficientAllowance
ERC20InvalidApprover
_msgSender
approve
_approve
ERC20InvalidSpender
Initializable
reinitializer
initializer
InvalidInitialization
NotInitializing
onlyInitializing
_checkInitializing
_disableInitializers
type
_isInitializing
_getInitializableStorage
_getInitializedVersion
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

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 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.

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