Audit Report

Borg Original

November 2023

Network GOERLI

Address 0x83c47769490cad939a50a75ded6c6d70bdbdb2be

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
●	TSD	Total Supply Diversion	Unresolved
●	UPA	Unexcluded Pinksale Address	Unresolved
●	RFM	Redundant Fee Mechanism	Unresolved
●	MAU	Misleading Address Usage	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	BorgOriginal
Testing Deploy	https://goerli.etherscan.io/address/0x83c47769490cad939a50a75ded6c6d70bdbdb2be
Symbol	BORG
Decimals	18
Total Supply	70,000,000,000,000

Audit Updates

Initial Audit	25 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/borg/v1/audit.pdf
Corrected Phase 2	02 Nov 2023 https://github.com/cyberscope-io/audits/blob/main/borg/v2/audit.pdf
Corrected Phase 3	06 Nov 2023

Source Files

Filename	SHA256
BorgOriginal.sol	d066ba0e4fefbc627aed2572bc22bd9df308cb0bb925c6c68935ce34edd0c2e2

Findings Breakdown

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	1	0	0	0
Minor / Informative	8	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	BorgOriginal.sol#L365
Status	Unresolved

Description

The contract is designed to calculate the `netAmount` variable, which represents the tax amount based on either the `sellTax` or the `totalFee` applied. However, instead of sending to the `recipient` the `amount` transferred minus the `netAmount` (which represents the fee), the contract is setting the balance of the `recipient` by adding the `netAmount`. As a result, if the tax fee is set up to 20%, the recipient receives only up to 20% of the transferred amount. This leads to an effective fee of 80%, which exceeds increase over the allowed limit of 25%.

```
 if (isSale) {
            netAmount = (amount * sellTax) / 100;
            totalFee = sellTax;
        } else {
            totalFee = reflectionFee1 + reflectionFee2 +
marketingFee +
            treasuryFee + liquidityFee + burnFee;
            netAmount = (amount * totalFee) / 100;
        }
        ...

        // Deduct the fee and send the net amount
        _balances[sender] -= amount;
        _balances[recipient] += netAmount;
```

Recommendation

The contract could embody a check for the maximum acceptable value. It is recommended to modify the contract's logic such that, instead of adding the `netAmount` to the `_balances` of the `recipient`, the contract should add `amount - netAmount`.

This results to the total transferred amount minus the fees, ensuring that the recipient receives the correct amount after tax deductions.

TSD - Total Supply Diversion

Criticality	Critical
Location	BorgOriginal.sol#L350
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

Specifically, the contract allocates various fees, including the `treasuryFeeShare` , from the total transaction amount. However, the contract attempts to calculate the `treasuryFeeShare` variable by subtracting only the fee variables from the total amount. This approach does not account for the `_balances` of the recipient. As a result, the balance of the `treasuryFeeShare` will be set to an incorrect number.

```
bool isSale = isExchangeAddress[recipient];
        if (isSale) {
            netAmount = (amount * sellTax) / 100;
            totalFee = sellTax;
        } else {
            totalFee = reflectionFee1 + reflectionFee2 + marketingFee +
            treasuryFee + liquidityFee + burnFee;
            netAmount = (amount * totalFee) / 100;
        }

        // Allocate liquidityFee and burnFee to their respective
destinations
        _balances[liquidityWallet] = _balances[liquidityWallet] +
(amount * liquidityFee) / 100;
        _balances[burnWallet] = _balances[burnWallet] + (amount *
burnFee) / 100;

        // Deduct the fee and send the net amount
        _balances[sender] -= amount;
        _balances[recipient] += netAmount;

        // Calculate all fees first to mitigate rounding errors
        uint256 reflectionFee1Share = (amount * reflectionFee1) / 100;
        uint256 liquidityFeeShare = (amount * liquidityFee) / 100;
        uint256 burnFeeShare = (amount * burnFee) / 100;
        uint256 reflectionFee2Share = (amount * reflectionFee2) / 100;
        uint256 marketingFeeShare = (amount * marketingFee) / 100;
        uint256 treasuryFeeShare = amount - (reflectionFee1Share +
reflectionFee2Share + marketingFeeShare + liquidityFeeShare +
burnFeeShare);

        _balances[reflectionAddress1] = _balances[reflectionAddress1] +
reflectionFee1Share;
        _balances[reflectionAddress2] = _balances[reflectionAddress2] +
reflectionFee2Share;
        _balances[marketingWallet] = _balances[marketingWallet] +
marketingFeeShare;
        _balances[treasuryWallet] = _balances[treasuryWallet] +
treasuryFeeShare;
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply. It is recommended to reconsider the calculation of the `treasuryFeeShare` fee. The fee should be calculated as the total amount minus the sum of all the fees that have occurred, plus the balance of the recipient. This will ensure that the `treasuryFeeShare` accurately represents the remaining fee after all other fees and balances have been accounted for.

UPA - Unexcluded Pinksale Address

Criticality	Medium
Location	BorgOriginal.sol#L315
Status	Unresolved

Description

The contract is designed with a fee mechanism that applies to each transaction. This design poses a significant obstacle to integration with platforms like Pinksale. Specifically, for the creation of a launchpad on Pinksale, the Pinksale factory address must be exempted from these transaction fees. Without this exemption, the creation of the pool on Pinksale will be prevented. Specifically, the contract does not define any actual functionality within the `if (applyFee)` block. This means that regardless of whether `applyFee` evaluates to true or false, no code is executed to actually apply or bypass fees, leading to a non-functional fee mechanism.

```
 bool applyFee = !_isExcludedFromFee[sender] &&
!_isExcludedFromFee[recipient];
        if (applyFee) {
        }
```

Recommendation

It is recommended to modify the contract to exclude the Pinksale factory address from the fee mechanism. This will ensure compatibility with Pinksale and facilitate the smooth creation of pools on the platform.

RFM - Redundant Fee Mechanism

Criticality	Minor / Informative
Location	BorgOriginal.sol#L350
Status	Unresolved

Description

The contract utilizes a set of constant fee variables: `reflectionFee1` , `reflectionFee2` , `marketingFee` , `treasuryFee` , `liquidityFee` , and `burnFee` , which collectively amount to a total fee of 20%. This total is equivalent to the `sellTax` variable, which is also set to 20%. The contract's logic includes a condition `if (isSale)` that applies the `sellTax` to calculate the `netAmount` and `totalFee` for sale transactions. However, this implementation introduces redundancy since the sum of the individual fee variables is equal to the `sellTax`. As a result, the constant fee variables could be used to distribute the fees instead of the `sellTax` variable.

```
uint8 public constant reflectionFee1 = 4; // 4% Reflection fee to first
mechanism
uint8 public constant reflectionFee2 = 4; // 4% Reflection fee to second
mechanism
uint8 public constant marketingFee = 4; // 4% Marketing fee
uint8 public constant treasuryFee = 4; // 4% Treasury fee
uint8 public constant liquidityFee = 3; // 3% Liquidity fee
uint8 public constant burnFee = 1; // 1% Burn fee
uint8 public constant sellTax = 20; // 20% Sell tax
 ...
 bool isSale = isExchangeAddress[recipient];
        if (isSale) {
            netAmount = (amount * sellTax) / 100;
            totalFee = sellTax;
        } else {
            totalFee = reflectionFee1 + reflectionFee2 + marketingFee +
            treasuryFee + liquidityFee + burnFee;
            netAmount = (amount * totalFee) / 100;
        }

        // Allocate liquidityFee and burnFee to their respective
destinations
        _balances[liquidityWallet] = _balances[liquidityWallet] + (amount
* liquidityFee) / 100;
        _balances[burnWallet] = _balances[burnWallet] + (amount *
burnFee) / 100;
```

Recommendation

It is recommended to refactor the contract's fee mechanism to eliminate the `sellTax` variable and rely solely on the sum of the individual fee variables. This approach will ensure consistency in fee application and simplify the contract's logic, making it more transparent and easier to maintain.

MAU - Misleading Address Usage

Criticality	Minor / Informative
Location	BorgOriginal.sol#L414
Status	Unresolved

Description

The contract contains a variable called `burnWallet` address to represent a specific type of address, commonly acknowledged in the blockchain for a particular purpose. However, this wallet address within this contract is mutable, meaning it can be altered. As a result, the designated address may not consistently serve its conventional purpose, potentially leading to unintended behaviors within the contract's operation. This mutable design diverges from the standard practice of utilizing a fixed, immutable address for such purposes, thereby introducing a layer of complexity and potential risk in the contract's functionality.

```
    function setBurnWallet(address _address) public onlyOwner {
          require(_address != address(0), "Cannot be zero
address");
          burnWallet = _address;
     }
```

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 clarity and comprehensibility of the code to ensure that it accurately reflects the intended functionality. The designated address, which reflects a specific purpose within the contract, should ideally be immutable to maintain consistency in its functionality and to adhere to common practices, thereby reducing the potential for unexpected behaviors or vulnerabilities within the contract's operation.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	BorgOriginal.sol#L392
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 excludeFromFee(address account) public onlyOwner {
        _isExcludedFromFee[account] = true;
    }

    function includeInFee(address account) public onlyOwner {
        _isExcludedFromFee[account] = false;
    }

    function isExcludedFromFee(address account) public view
returns(bool) {
        return _isExcludedFromFee[account];
    }

    function setReflectionAddress1(address _address) public onlyOwner {
        require(_address != address(0), "Cannot be zero address");
        reflectionAddress1 = _address;
    }

    function setLiquidityWallet(address _address) public onlyOwner {
        require(_address != address(0), "Cannot be zero address");
        liquidityWallet = _address;
    }

    function setBurnWallet(address _address) public onlyOwner {
        require(_address != address(0), "Cannot be zero address");
        burnWallet = _address;
    }

    function setExchangeAddress(address _address, bool _status) public
onlyOwner {
        require(_address != address(0), "Cannot be zero address");
        isExchangeAddress[_address] = _status;
    }
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BorgOriginal.sol#L248
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.

```
address public wbnbAddress =
0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
```

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	BorgOriginal.sol#L404,409,414,419
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.

```
address _address
bool _status
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	BorgOriginal.sol#L6,33,135,215
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.20;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BorgOriginal.sol#L6,33,135,215
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;
pragma solidity ^0.8.0;
```

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	BorgOriginal.sol#L252,253
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.

```
IERC20(wbnbAddress).transfer(marketingWallet, marketingAmount)
IERC20(wbnbAddress).transfer(treasuryWallet, treasuryAmount)
```

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.

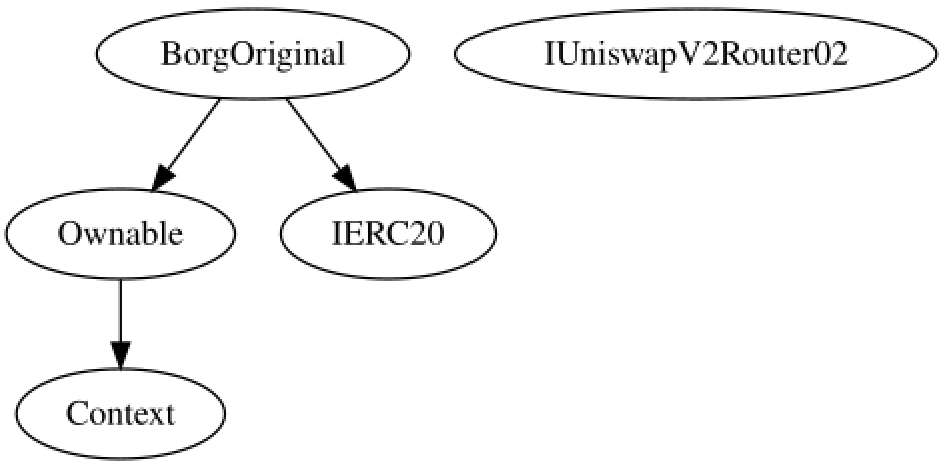
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-

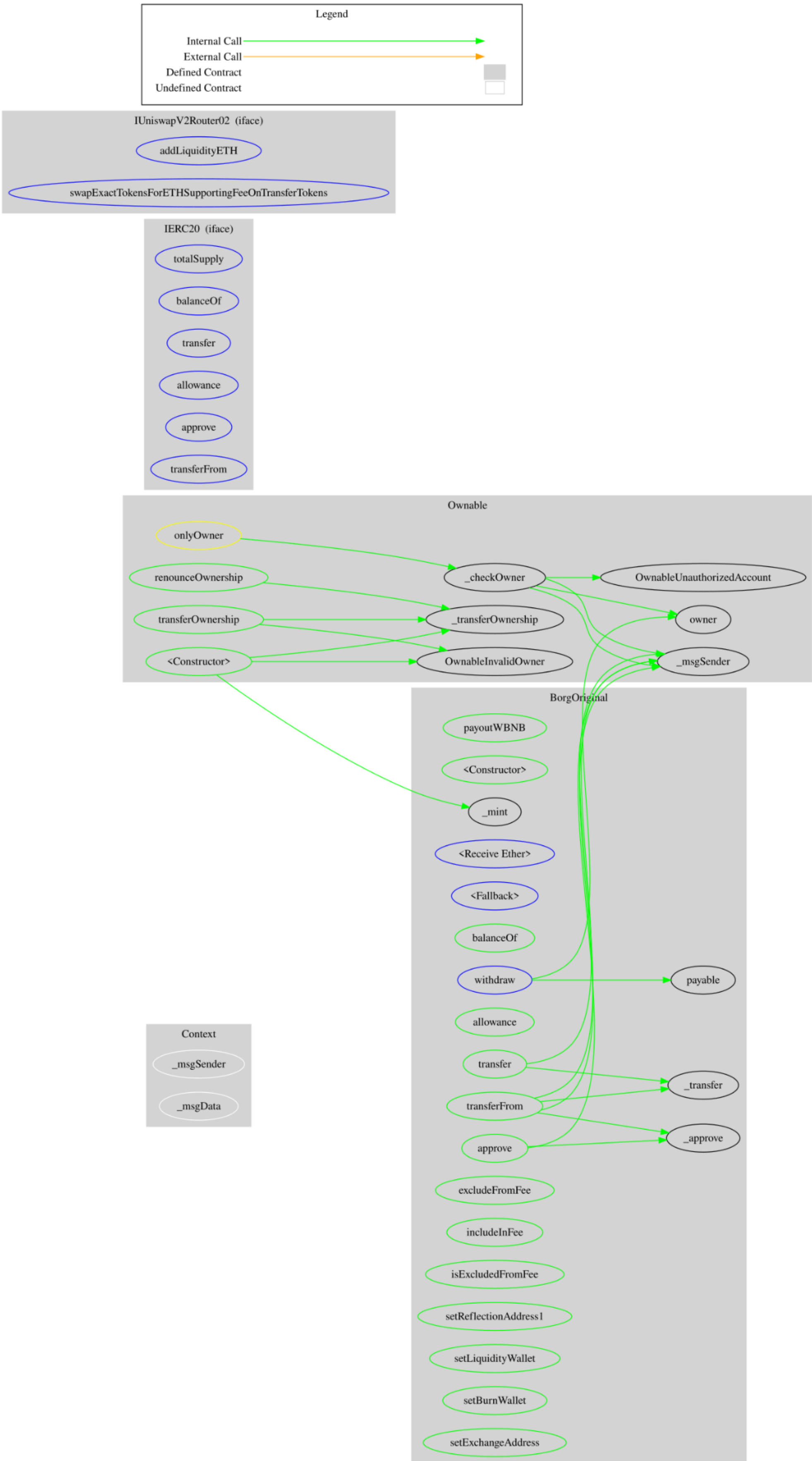
	transferFrom	External	✓	-
IUniswapV2Router02	Interface			
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
BorgOriginal	Implementation	IERC20, Ownable		
	payoutWBNB	Public	✓	onlyOwner
		Public	✓	Ownable
	_mint	Internal	✓	
		External	Payable	-
		External	✓	-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Internal	✓	
	_transfer	Internal	✓	
	withdraw	External	✓	onlyOwner
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	isExcludedFromFee	Public		-

	setReflectionAddress1	Public	✓	onlyOwner
	setLiquidityWallet	Public	✓	onlyOwner
	setBurnWallet	Public	✓	onlyOwner
	setExchangeAddress	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router02 (iface)
addLiquidityETH
swapExactTokensForETHSupportingFeeOnTransferTokens
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
renounceOwnership
transferOwnership
<Constructor>
_checkOwner
OwnableUnauthorizedAccount
_transferOwnership
owner
OwnableInvalidOwner
_msgSender
BorgOriginal
payoutWBNB
<Constructor>
_mint
<Receive Ether>
<Fallback>
balanceOf
withdraw
payable
allowance
transfer
_transfer
transferFrom
approve
_approve
excludeFromFee
includeInFee
isExcludedFromFee
setReflectionAddress1
setLiquidityWallet
setBurnWallet
setExchangeAddress
Context
_msgSender
_msgData

Summary

Borg Original 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