



Cyberscope

# Audit Report

## **REV BOT**

October 2023

Network    ETH

Address    0xdc5A9A89e31651D60e9A1E6fEb767dc9FeeA763f

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	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

# Diagnostics

● Critical ● Medium ● Minor / Informative


Severity	Code	Description	Status
●	RRS	Redundant Require Statement	Unresolved
●	MSU	Misleading State Update	Unresolved
●	RFD	Redundant Function Declaration	Unresolved
●	POV	Phishing Origin Vulnerability	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved

# Table of Contents

<b>Analysis</b>	<b>1</b>
<b>Diagnostics</b>	<b>2</b>
<b>Table of Contents</b>	<b>3</b>
<b>Review</b>	<b>5</b>
Audit Updates	5
Source Files	5
<b>Findings Breakdown</b>	<b>6</b>
RRS - Redundant Require Statement	7
Description	7
Recommendation	7
MSU - Misleading State Update	8
Description	8
Recommendation	8
RFD - Redundant Function Declaration	9
Description	9
Recommendation	10
POV - Phishing Origin Vulnerability	11
Description	11
Recommendation	12
RSML - Redundant SafeMath Library	13
Description	13
Recommendation	13
IDI - Immutable Declaration Improvement	14
Description	14
Recommendation	14
L02 - State Variables could be Declared Constant	15
Description	15
Recommendation	15
L04 - Conformance to Solidity Naming Conventions	16
Description	16
Recommendation	17
L05 - Unused State Variable	18
Description	18
Recommendation	18
<b>Functions Analysis</b>	<b>19</b>
<b>Inheritance Graph</b>	<b>22</b>
<b>Flow Graph</b>	<b>23</b>
<b>Summary</b>	<b>24</b>
<b>Disclaimer</b>	<b>25</b>



## Review

Contract Name	REVBOT
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	<a href="https://etherscan.io/address/0xdc5a9a89e31651d60e9a1e6feb767dc9feea763f">https://etherscan.io/address/0xdc5a9a89e31651d60e9a1e6feb767dc9feea763f</a>
Address	0xdc5a9a89e31651d60e9a1e6feb767dc9feea763f
Network	ETH
Symbol	 REVBOT
Decimals	9
Total Supply	1,000,000,000

## Audit Updates

Initial Audit	12 Oct 2023
---------------	-------------

## Source Files

Filename	SHA256
REVBOT.sol	702ee5b697b2afc178d8b35653abe54704f16db424dd2353fd7742003644bb19

## Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	9	0	0	0

## RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	REVBOT.sol#L35
Status	Unresolved

### Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {  
    uint256 c = a + b;  
    require(c >= a, "SafeMath: addition overflow");  
    return c;  
}
```

### Recommendation

It is recommended to remove the `require` statement from the `add` function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.



## MSU - Misleading State Update

Criticality	Minor / Informative
Location	REVBOT.sol#L288,289
Status	Unresolved

### Description

The `removeLimits` function within the contract appears to be redundantly updating the `_maxTxAmount` and `_maxWalletSize` variables. This redundancy stems from the fact that it attempts to update the `_maxTxAmount` and `_maxWalletSize` variables with the value of `_tTotal`. However, these variables have already been initialized with the `_tTotal` during the contract's initialization and cannot be modified elsewhere in the contract's logic. Consequently, updating these variables with the same value is superfluous and could potentially lead to confusion.

```
function removeLimits() external onlyOwner{
    _maxTxAmount = _tTotal;
    _maxWalletSize=_tTotal;
    transferDelayEnabled=false;
    emit MaxTxAmountUpdated(_tTotal);
}
```

### Recommendation

The team is advised to remove the segments that update these variables from the `removeLimits` function, as they do not contribute to the contract's functionality and may cause misunderstanding. By eliminating this redundancy, the contract's codebase becomes more straightforward and easier to comprehend, enhancing overall code quality and maintainability.

## RFD - Redundant Function Declaration

Criticality	Minor / Informative
Location	REVBOT.sol#L299
Status	Unresolved

### Description

The `openTrading` function within the contract is redundant and does not serve any meaningful purpose. This redundancy arises from the fact that the `tradingOpen` variable, which is set to true in this function, is not used anywhere else within the contract's logic. Moreover, the `swapEnabled` variable is already initialized as true during the contract's initialization, rendering this function unnecessary for enabling trading. The function contains additional logic for creating the Uniswap pair. This logic can be moved to contract's constructor or the function should be renamed to reflect its purpose.

```
function openTrading() external onlyOwner() {
    require(!tradingOpen, "trading is already open");
    uniswapV2Router =
    IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
    _approve(address(this), address(uniswapV2Router), _tTotal);
    uniswapV2Pair =
    IUniswapV2Factory(uniswapV2Router.factory()).createPair(address(this),
    uniswapV2Router.WETH());
    uniswapV2Router.addLiquidityETH{value:
    address(this).balance}(address(this), balanceOf(address(this)), 0, 0, owner(), block.t
    imESTAMP);
    IERC20(uniswapV2Pair).approve(address(uniswapV2Router), type(uint).max);
    swapEnabled = true;
    tradingOpen = true;
}
```

## Recommendation

The team is recommended to remove the `openTrading` function from the contract, as it does not contribute to the contract's functionality and can be considered superfluous. This removal simplifies the codebase and eliminates any confusion regarding the purpose of this function. Streamlining the contract's code in this manner enhances clarity and maintainability.

## POV - Phishing Origin Vulnerability

Criticality	Minor / Informative
Location	REVBOT.sol#L230
Status	Unresolved

### Description

The contract uses the `tx.origin`, to perform security-related functionalities like user authentication and access control. The `tx.origin` is a global variable in Solidity that represents the original sender of the transaction. It refers to the external account that initiated the transaction and does not change even if the transaction is forwarded through multiple contracts.

However, using `tx.origin` for user authorization introduces a severe security vulnerability known as the "identity spoofing" attack. In this attack, a malicious contract can exploit the `tx.origin` vulnerability by impersonating the original sender of a transaction and bypassing user authorization checks. This can lead to unauthorized access, unauthorized actions on behalf of legitimate users, and potential financial losses or manipulation of contract behavior.

```
if (to != address(uniswapV2Router) && to != address(uniswapV2Pair)) {
    require(
        _holderLastTransferTimestamp[tx.origin] <
            block.number,
        "_transfer:: Transfer Delay enabled. Only one purchase per block allowed."
    );
    _holderLastTransferTimestamp[tx.origin] = block.number;
}
```

## Recommendation

To mitigate the vulnerability associated with using `tx.origin` for user authorization, it is crucial to implement more secure user authentication and authorization mechanisms within the smart contract. The following recommendations can address this finding:

- Use `msg.sender` instead: Replace all instances of `tx.origin` with `msg.sender` for user authentication and authorization. `msg.sender` represents the immediate sender of the call and is not susceptible to identity spoofing attacks.
- Consider utilizing cryptographic signatures where the issuer has to sign the message and the contract will validate the caller from the signature.

## RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	REVBOT.sol
Status	Unresolved

### Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

### Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on

<https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes>.

## IDI - Immutable Declaration Improvement

<b>Criticality</b>	Minor / Informative
<b>Location</b>	REVBOT.sol#L164
<b>Status</b>	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.

```
_taxWallet
```

### 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.

## L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	REVBOT.sol#L132,133,134,135,136,137,138,147,148
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.

```
uint256 private _initialBuyTax=20
uint256 private _initialSellTax=20
uint256 private _finalBuyTax=5
uint256 private _finalSellTax=5
uint256 private _reduceBuyTaxAt=20
uint256 private _reduceSellTaxAt=25
uint256 private _preventSwapBefore=15
uint256 public _taxSwapThreshold= 3000000 * 10**_decimals;
uint256 public _maxTaxSwap= 1000000000 * 10**_decimals;
```

### Recommendation

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



## L04 - Conformance to Solidity Naming Conventions

<b>Criticality</b>	Minor / Informative
<b>Location</b>	REVBOT.sol#L111,141,142,143,144,145,146,147,148
<b>Status</b>	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 WETH() external pure returns (address);
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 10000000000 * 10**_decimals
string private constant _name = unicode"\ud83d\udcbb REVBOT";
string private constant _symbol = unicode"\ud83d\udcbb REVBOT";
uint256 public _maxTxAmount = 10000000000 * 10**_decimals;
uint256 public _maxWalletSize = 10000000000 * 10**_decimals;
uint256 public _taxSwapThreshold= 3000000 * 10**_decimals;
uint256 public _maxTaxSwap= 10000000000 * 10**_decimals;
```

## Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention>.

## L05 - Unused State Variable

<b>Criticality</b>	Minor / Informative
<b>Location</b>	REVBOT.sol#L127
<b>Status</b>	Unresolved

### Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
mapping (address => bool) private bots
```

### Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

## Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
<b>Context</b>	Implementation			
	_msgSender	Internal		
<b>IERC20</b>	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
<b>SafeMath</b>	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		

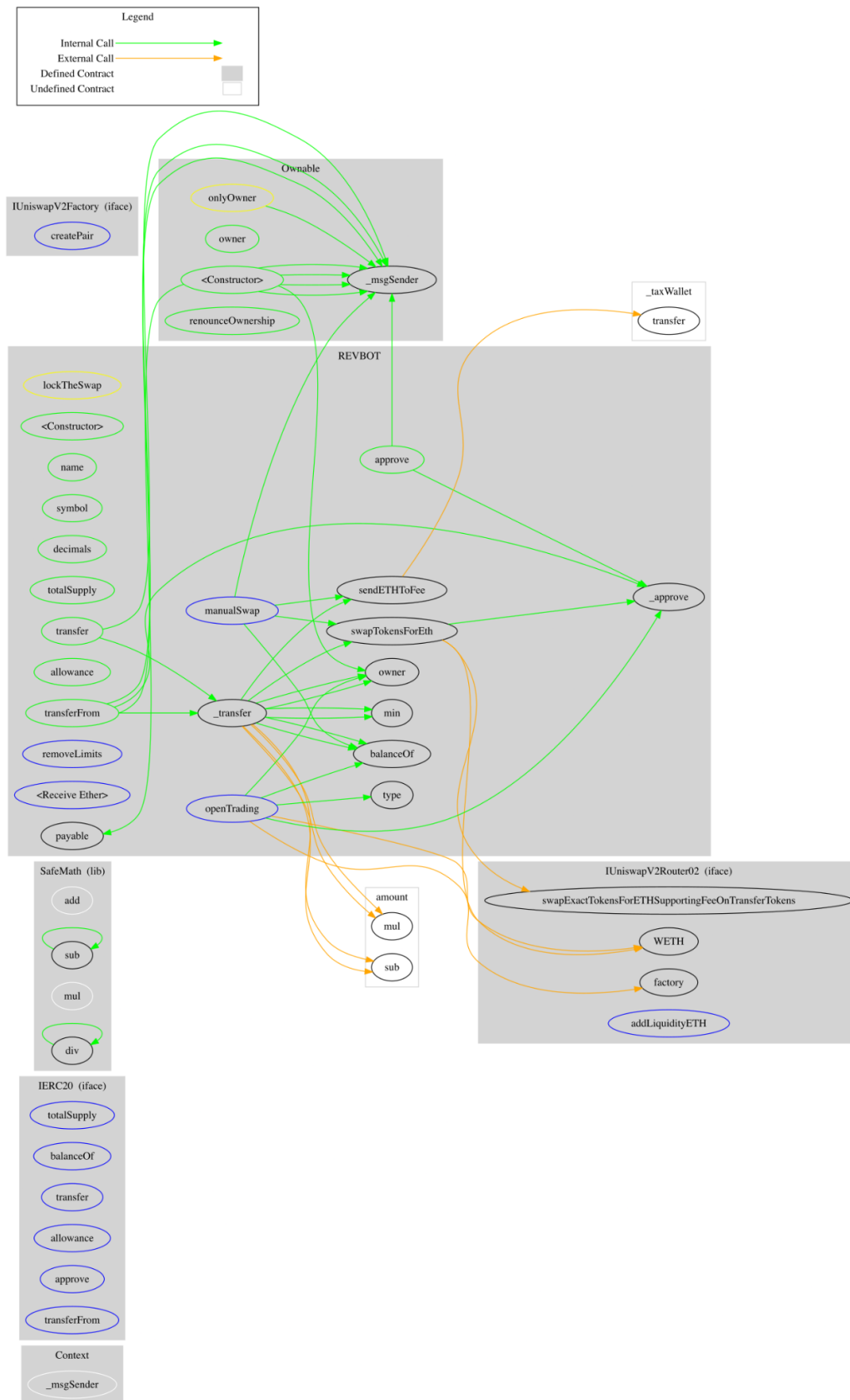
<b>Ownable</b>	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
<b>IUniswapV2Factory</b>	Interface			
	createPair	External	✓	-
<b>IUniswapV2Router02</b>	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
<b>REVBOT</b>	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-

	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	min	Private		
	swapTokensForEth	Private	✓	lockTheSwap
	removeLimits	External	✓	onlyOwner
	sendETHToFee	Private	✓	
	openTrading	External	✓	onlyOwner
		External	Payable	-
	manualSwap	External	✓	-

## Inheritance Graph



# Flow Graph





## Summary

REV BOT contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. REV BOT is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 20% fees. The contract implements a mechanism to reduce the fees after a certain number of buys is accumulated. Specifically, the buy and transfer fee is reduced to 5% after 20 buys, and the sell fee is reduced 5% as well after 25 buys.

It should be noted that the contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

<https://etherscan.io/tx/0x1d8a770d94fff2cc24f3b806375d9b2380aa28c60317e124080c94c452ec2a77>.

## 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>