Cyberscope

# Audit Report

# AI Minion

April 2023

| | |
|---|---|
| Network | BSC |
| Address | 0x2AB657B29778621A723667DFf60Df0b614c35b98 |
| Audited by | © cyberscope |

# Table of Contents

# Review

| | |
|---|---|
| **Contract Name** | AIMINION |
| **Compiler Version** | v0.8.18+commit.87f61d96 |
| **Optimization** | 200 runs |
| **Explorer** | https://bscscan.com/address/0x2ab657b29778621a723667dff60df0b614c35b98 |
| **Address** | 0x2ab657b29778621a723667dff60df0b614c35b98 |
| **Network** | BSC |
| **Symbol** | AIM |
| **Decimals** | 9 |
| **Total Supply** | 24,999,999,999,999 |

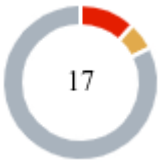
## Audit Updates

| | |
|---|---|
| **Initial Audit** | 10 Apr 2023 |

## Source Files

| | |
|---|---|
| **Filename** | SHA256 |
| **AIMINION.sol** | a1bbe0ef539a9592c5e5a0b53eb7d9324bf302190fe6924696815fdaa7000d93 |

# Findings Breakdown




| | |
|---|---|
| Critical | 2 |
| Medium | 1 |
| Minor / Informative | 14 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 2 | 0 | 0 | 0 |
| Medium | 1 | 0 | 0 | 0 |
| Minor / Informative | 14 | 0 | 0 | 0 |

# Analysis

Critical Medium Minor / Informative Pass

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | ST | Stops Transactions | Unresolved |
| ● | OCTD | Transfers Contract's Tokens | Passed |
| ● | OTUT | Transfers User's Tokens | Passed |
| ● | ELFM | Exceeds Fees Limit | Passed |
| ● | ULTW | Transfers Liquidity to Team Wallet | Passed |
| ● | MT | Mints Tokens | Passed |
| ● | BT | Burns Tokens | Passed |
| ● | BC | Blacklists Addresses | Unresolved |

## ST - Stops Transactions

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | AIMINION.sol#L1025,1153 |
| **Status** | Unresolved |

### Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `gasPriceLimit` to zero.

Additionally, the owner has the authority to mark users as `snipers` during sales, by manipulating the `snipeBlockAmt` variable. As a result, the contract may operate as a honeypot.

```
if (gasLimitActive) {
    require(tx.gasprice <= gasPriceLimit, "Gas price exceeds limit.");
}
...
if (block.number - _liqAddBlock < snipeBlockAmt) {
    _isSniper[to] = true;
    snipersCaught ++;
    emit SniperCaught(to);
}
```

### Recommendation

The contract could embody a check for not allowing setting the gasPriceLimit less than a reasonable amount. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

## BC - Blacklists Addresses

| | |
|---|---|
| **Criticality** | Medium |
| **Location** | AIMINION.sol#L1139 |
| **Status** | Unresolved |

### Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by enabling the `sniperProtection` variable.

```
if (isSniper(from) || isSniper(to)) {
    revert("Sniper rejected.");
}
```

### 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

# Diagnostics

Critical Medium Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| Critical | CLF | Contract Locked Funds | Unresolved |
| Minor / Informative | PTRP | Potential Transfer Revert Propagation | Unresolved |
| Minor / Informative | RC | Redundant Check | Unresolved |
| Minor / Informative | RDM | Revert Descriptive Message | Unresolved |
| Minor / Informative | RA | Redundant Allowance | Unresolved |
| Minor / Informative | IDI | Immutable Declaration Improvement | Unresolved |
| Minor / Informative | L02 | State Variables could be Declared Constant | Unresolved |
| Minor / Informative | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| Minor / Informative | L05 | Unused State Variable | Unresolved |
| Minor / Informative | L07 | Missing Events Arithmetic | Unresolved |
| Minor / Informative | L09 | Dead Code Elimination | Unresolved |
| Minor / Informative | L11 | Unnecessary Boolean equality | Unresolved |
| Minor / Informative | L13 | Divide before Multiply Operation | Unresolved |
| Minor / Informative | L14 | Uninitialized Variables in Local Scope | Unresolved |

| | | | |
|---|---|---|---|
| ● | L17 | Usage of Solidity Assembly | Unresolved |

## CLF - Contract Locked Funds

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | AIMINION.sol#L1070 |
| **Status** | Unresolved |

### Description

The `swapAndLiquify` function does not swap the whole `contractTokenBalance`. A portion of the `contractTokenBalance` remains at the contract. The contract does not implement any function to retrieve these funds. As a result, the remaining tokens will stay locked at the contract.

```
function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap
{}
```

### Recommendation

The team is advised to take these segments into consideration and rewrite them to avoid locking funds at the contract. A recommendation would be to fork the `swapAndLiquify` function from well-known contracts.

# PTRP - Potential Transfer Revert Propagation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L1096,1097 |
| **Status** | Unresolved |

## Description

The contract sends funds to a `marketingWallet` and a `charityWallet` as part of the transfer flow. These addresses 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.

```
_marketingWallet.transfer(amountMarketingBNB);
_charityWallet.transfer(amountCharityBNB);
```

## 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 achived by not allowing set contract addresses or by sending the funds in a non-revertable way.

# RC - Redundant Check

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L856,874,892 |
| **Status** | Unresolved |

## Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract functions `setBuyTaxes` , `setSellTaxes` and `setTransferTaxes` require that each fee is less than or equal to a maximum value, and their sum is less than 5000. The sum of the maximum values of each fee is equal to 2000, so even if each fee is given the maximum acceptable value, the sum of all fees will always be less thn 5000. As a result, the second `require` statement is redundant.

```
require(reflectFee <= maxReflectFee
    && marketingFee <= maxMarketingFee
    && charityFee <= maxCharityFee
    && burnFee <= maxBurnFee);
require(reflectFee + marketingFee + charityFee + burnFee <= 5000);
```

## Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

## RDM - Revert Descriptive Message

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L852,856,870,874,888,892,1031,1034 |
| **Status** | Unresolved |

### Description

The `require()` function is used to halt the execution of a contract, if a certain condition is not met, and revert any changes made to the contract's state. The contract does not provide a descriptive message to the `require()` function.

```
require(reflectFee <= maxReflectFee
    && marketingFee <= maxMarketingFee
    && charityFee <= maxCharityFee
    && burnFee <= maxBurnFee);
require( reflectFee + marketingFee+ charityFee + burnFee <= 5000);
require(lastTrade[to] != block.number);
```

### Recommendation

The team is suggested to provide a descriptive message to the `require()` function. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

# RA - Redundant Allowance

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L726 |
| **Status** | Unresolved |

## Description

The `transferOwner` function transfers the ownership of the contract to a new owner. The previous owner's balance is transferred to the new owner. Additionally, the function grants an allowance equal to the owner's balance for the new owner. Since all the balance of the owner is transferred, setting a new allowance between the two addresses is redundant.

```
_allowances[_owner][newOwner] = balanceOf(_owner);
if(balanceOf(_owner) > 0) {
    _transfer(_owner, newOwner, balanceOf(_owner));
}
```

## Recommendation

The team is advised to remove the code segment where the allowance is being set.

## IDI - Immutable Declaration Improvement

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L671 |
| **Status** | Unresolved |

### Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
lpPairObj
```

### 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** | AIMINION.sol#L538,541,547,548,570,571,572,573,575,587,589,592,598,599,604,605,624,626 |
| **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 startingSupply = 25_000_000_000_000
uint8 private _decimals = 9
string private _name = "AI Minion"
string private _symbol = "AIM"
uint256 private maxReflectFee = 800
uint256 private maxMarketingFee = 600
uint256 private maxCharityFee = 300
uint256 private maxBurnFee = 300
uint256 private masterTaxDivisor = 10000
address private _routerAddress =
0x10ED43C718714eb63d5aA57B78B54704E256024E
address public burnAddress = 0x000000000000000000000000000000000000dEaD
address payable private _charityWallet =
payable(0xB1C0CF66a8506302A7aBa1640A3a9A249E697372)
uint256 private maxTxPercent = 5
uint256 private maxTxDivisor = 100

...
```

## 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** | AIMINION.sol#L351,352,366,384,550,551,552,553,555,556,557,558,560,561,562,563,565,566,567,568,613,840,919 |
| **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);
uint256 public _reflectFee = 0
uint256 public _marketingFee = 0
uint256 public _charityFee = 0
uint256 public _burnFee = 0
uint256 public _buyReflectFee = 100
uint256 public _buyMarketingFee = 100
uint256 public _buyCharityFee = 100
uint256 public _buyBurnFee = 100
uint256 public _sellReflectFee = _buyReflectFee
uint256 public _sellMarketingFee = _buyMarketingFee

...
```

## 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

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L577,578,579,580,601,607,624 |
| **Status** | 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.

```
uint256 private _previousReflectFee = _reflectFee
uint256 private _previousMarketingFee = _marketingFee
uint256 private _previousCharityFee = _charityFee
uint256 private _previousBurnFee = _burnFee
uint256 private _previousMaxTxAmount = _maxTxAmount
uint256 private _previousMaxWalletSize = _maxWalletSize
address private lastSell
```

## 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.

## L07 - Missing Events Arithmetic

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L825,842,858,876,894,903,910 |
| **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.

```
_liqAddStatus = rInitializer
snipeBlockAmt = _block
_buyReflectFee = reflectFee
_sellReflectFee = reflectFee
_transferReflectFee = reflectFee
_maxTxAmount = check
_maxWalletSize = check
```

### 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** | AIMINION.sol#L265,276,284,288,293,297,302 |
| **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 isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet
created accounts
        // and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

...
```

## 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** | AIMINION.sol#L796,925,932 |
| **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.

```
enabled == false
enabled == true
```

### 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** | AIMINION.sol#L1077,1085 |
| **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 a loss of prediction.

```
uint256 half = toLiquify / 2
uint256 liquidityBalance = (fromSwap * half) / toSwapForEth
```

### 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.

## L14 - Uninitialized Variables in Local Scope

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L1198 |
| **Status** | Unresolved |

### Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
ExtraValues memory values
```

### Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

## L17 - Usage of Solidity Assembly

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | AIMINION.sol#L272,315 |
| **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 { codehash := extcodehash(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

### 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.

# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **Context** | Implementation | | | |
| | _msgSender | Internal | | |
| | _msgData | Internal | | |
| | | | | |
| **IERC20** | Interface | | | |
| | totalSupply | External | | - |
| | decimals | External | | - |
| | symbol | External | | - |
| | name | External | | - |
| | getOwner | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **SafeMath** | Library | | | |
| | add | Internal | | |

| | | | | |
|---|---|---|---|---|
| | sub | Internal | | |
| | sub | Internal | | |
| | mul | Internal | | |
| | div | Internal | | |
| | div | Internal | | |
| | mod | Internal | | |
| | mod | Internal | | |
| | | | | |
| **Address** | Library | | | |
| | isContract | Internal | | |
| | sendValue | Internal | ✓ | |
| | functionCall | Internal | ✓ | |
| | functionCall | Internal | ✓ | |
| | functionCallWithValue | Internal | ✓ | |
| | functionCallWithValue | Internal | ✓ | |
| | _functionCallWithValue | Private | ✓ | |
| | | | | |
| **IUniswapV2Factory** | Interface | | | |
| | feeTo | External | | - |
| | feeToSetter | External | | - |
| | getPair | External | | - |
| | allPairs | External | | - |
| | allPairsLength | External | | - |

| | | | | |
|---|---|---|---|---|
| | createPair | External | ✓ | - |
| | setFeeTo | External | ✓ | - |
| | setFeeToSetter | External | ✓ | - |
| | | | | |
| **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 | ✓ | - |
| | | | | |
| **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 | ✓ | - |

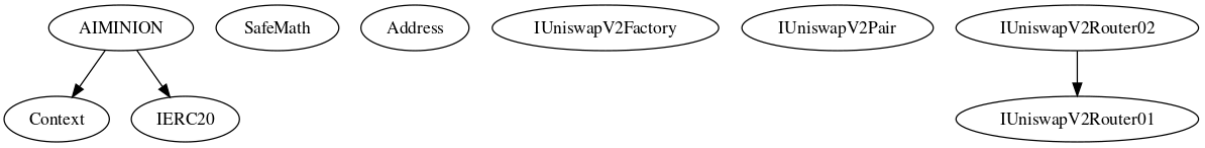
| | | | | |
|---|---|---|---|---|
| | swapExactTokensForETH | External | ✓ | - |
| | swapETHForExactTokens | External | Payable | - |
| | quote | External | | - |
| | getAmountOut | External | | - |
| | getAmountIn | External | | - |
| | getAmountsOut | External | | - |
| | getAmountsIn | External | | - |
| | | | | |
| **IUniswapV2Router02** | Interface | IUniswapV2 Router01 | | |
| | removeLiquidityETHSupportingFeeOnTransferTokens | External | ✓ | - |
| | removeLiquidityETHWithPermitSupportingFeeOnTransferTokens | External | ✓ | - |
| | swapExactTokensForTokensSupportingFeeOnTransferTokens | External | ✓ | - |
| | swapExactETHForTokensSupportingFeeOnTransferTokens | External | Payable | - |
| | swapExactTokensForETHSupportingFeeOnTransferTokens | External | ✓ | - |
| | | | | |
| **AIMINION** | Implementation | Context, IERC20 | | |
| | | Public | Payable | - |
| | | External | Payable | - |
| | owner | Public | | - |
| | transferOwner | External | ✓ | onlyOwner |
| | renounceOwnership | Public | ✓ | onlyOwner |
| | totalSupply | External | | - |

| | | | | |
|---|---|---|---|---|
| | decimals | External | | - |
| | symbol | External | | - |
| | name | External | | - |
| | getOwner | External | | - |
| | allowance | External | | - |
| | balanceOf | Public | | - |
| | transfer | Public | ✓ | - |
| | approve | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | setNewRouter | Public | ✓ | onlyOwner |
| | setLpPair | External | ✓ | onlyOwner |
| | isExcludedFromReward | Public | | - |
| | isExcludedFromFee | Public | | - |
| | isTransferTaxExcluded | Public | | - |
| | isSniper | Public | | - |
| | isProtected | External | ✓ | onlyOwner |
| | removeSniper | External | ✓ | onlyOwner |
| | setProtectionSettings | External | ✓ | onlyOwner |
| | setStartingProtections | External | ✓ | onlyOwner |
| | setBuyTaxes | External | ✓ | onlyOwner |
| | setSellTaxes | External | ✓ | onlyOwner |

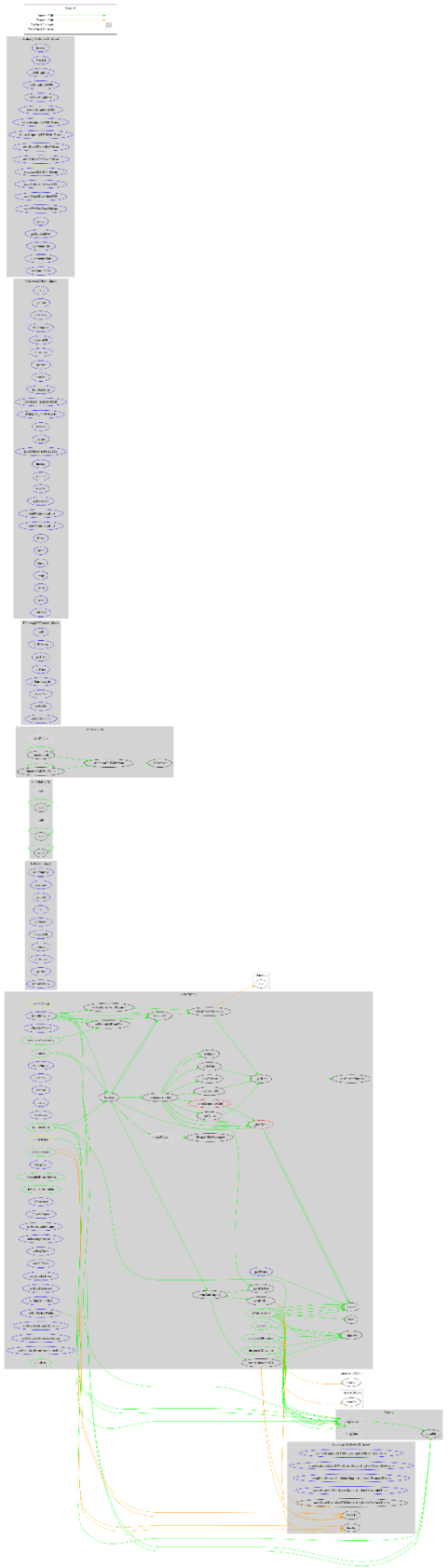
| | | | | |
|---|---|---|---|---|
| | setTransferTaxes | External | ✓ | onlyOwner |
| | setMaxTxPercent | External | ✓ | onlyOwner |
| | setMaxWalletSize | External | ✓ | onlyOwner |
| | setMarketingWallet | External | ✓ | onlyOwner |
| | setSwapAndLiquifyEnabled | External | ✓ | onlyOwner |
| | setExcludedFromReward | Public | ✓ | onlyOwner |
| | setExcludedFromFee | Public | ✓ | onlyOwner |
| | setExcludedFromTransferTax | External | ✓ | onlyOwner |
| | setExcludedFromTransferTaxBatch | External | ✓ | onlyOwner |
| | totalFees | Public | | - |
| | _hasLimits | Private | | |
| | tokenFromReflection | Public | | - |
| | _approve | Private | ✓ | |
| | adjustTaxes | Internal | ✓ | |
| | _transfer | Internal | ✓ | |
| | swapAndLiquify | Private | ✓ | lockTheSwap |
| | sendBNBout | Internal | ✓ | |
| | swapTokensForEth | Private | ✓ | |
| | _checkLiquidityAdd | Private | ✓ | |
| | _finalizeTransfer | Private | ✓ | |
| | getBNBFee | Internal | | |
| | _getValues | Private | | |
| | _getRate | Private | | |

| | | | | |
|---|---|---|---|---|
| | _getCurrentSupply | Private | | |
| | _takeReflect | Private | ✓ | |
| | _takeLiquidity | Private | ✓ | |
| | _takeBurn | Private | ✓ | |

# Inheritance Graph



# Flow Graph



# Summary

AI Minion 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 blacklist addresses. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. 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 20% buy, sell, and transfer 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