Cyberscope

Audit Report

Draggy

November 2024

Network ETH

Address 0xd12a99dbc40036cec6f1b776dccd2d36f5953b94

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
Minor / Informative	AOI	Arithmetic Operations Inconsistency	Unresolved
Minor / Informative	CO	Code Optimization	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	MEM	Missing Error Messages	Unresolved
Minor / Informative	NWES	Nonconformity with ERC-20 Standard	Unresolved
Minor / Informative	PGA	Potential Griefing Attack	Unresolved
Minor / Informative	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
Minor / Informative	RRA	Redundant Repeated Approvals	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	RSRS	Redundant SafeMath Require Statement	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved
Minor / Informative	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	Draggy
Compiler Version	v0.8.23+commit.f704f362
Optimization	200 runs
Explorer	https://etherscan.io/address/0xd12a99dbc40036cec6f1b776dccd2d36f5953b94
Address	0xd12a99dbc40036cec6f1b776dccd2d36f5953b94
Network	ETH
Symbol	DRAGGY
Decimals	9
Total Supply	420,690,000,000,000

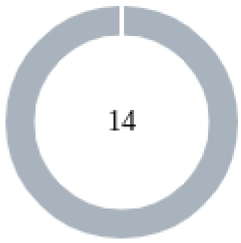
Audit Updates

Initial Audit	05 Nov 2024

Source Files

Filename	SHA256
Draggy.sol	d80944ab85cf64f298e24e2c37c3545cac513edf8c5b6c514c8edcf02e145a70

Findings Breakdown



- Critical 0
- Medium 0
- Minor / Informative 14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	Draggy.sol#L251,253,260,296,300
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
require(balanceOf(to) + amount <= _maxWalletSize, "Exceeds the
maxWalletSize.");
```

```
_balances[address(this)]=_balances[address(this)].add(taxAmount);
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

CO - Code Optimization

Criticality	Minor / Informative
Location	Draggy.sol#L247,264
Status	Unresolved

Description

The contract implements 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. Specifically, the contract's fee mechanism exhibits redundant complexity. It assumes three levels of tax application: an initial tax, a middle tax, and a final tax. However, the first two levels are initialized with identical parameters, such as `_initialBuyTax=200` and `_midBuyTax=200`, as well as `_initialSellTax=200` and `_midSellTax=200`. Consequently, the mid-tier is effectively neglected, and only the initial and final tiers are applicable. The following code segments therefore exhibit redundant complexity and can be optimized to improve contract readability and consistency.

```
taxAmount = amount.mul((_buyCount> _reduceBuyTaxAt)? _finalBuyTax:
((_buyCount> _midBuyTaxAt)? _midBuyTax: _initialBuyTax)).div(1000);
```

```
taxAmount = amount.mul((_buyCount> _reduceSellTaxAt)? _finalSellTax:
((_buyCount> _midSellTaxAt)? _midSellTax: _initialSellTax)).div(1000);
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Draggy.sol#L175
Status	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.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	Draggy.sol#L250,340,345,350
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(!isContract(to))
require(rescueSwitch || tradingOpen)
require(_msgSender()==_taxWallet)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. 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.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	Draggy.sol#L243
Status	Unresolved

Description

The contract is not fully conforming to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However the contract implements, a conditional check that prohibits transfers of 0 values.

This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
function _transfer(address from, address to, uint256 amount)
private {
...
require(amount > 0, "Transfer amount must be greater than zero");
...
}
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PGA - Potential Griefing Attack

Criticality	Minor / Informative
Location	Draggy.sol#L276
Status	Unresolved

Description

The `_transfer` function includes a `require` statement designed to limit the contract to a maximum of two sales per block. This setup opens the potential for a griefieng attack. During such an incident, a malicious actor could front-run normal sales with spam transactions, triggering the swap mechanism, to prevent legitimate transactions from selling tokens.

```
function _transfer(address from,address to,uint256 amount) private
{
...
if (!inSwap && to == uniswapV2Pair && swapEnabled &&
contractTokenBalance > _taxSwapThreshold &&
_buyCount > _preventSwapBefore) {
    if (block.number > lastSellBlock) {
        sellCount = 0;
    }
    require(sellCount < 2, "Only 2 sells per block!");
    swapTokensForEth(min(amount, min(contractTokenBalance,
_maxTaxSwap)));
...
}
```

Recommendation

The team is advised to review the transfer mechanism in regards to sales to ensure all legitimate transactions are processed according to the intended behavior. Decoupling the swap mechanism from the transaction's success could help resolve this issue.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Draggy.sol#L317
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForEth(uint256 tokenAmount) private lockTheSwap
{
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router), tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	Draggy.sol#L317
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
function swapTokensForEth(uint256 tokenAmount) private lockTheSwap
{
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = uniswapV2Router.WETH();
_approve(address(this), address(uniswapV2Router), tokenAmount);
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
tokenAmount,
0,
path,
address(this),
block.timestamp
);
}
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Draggy.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 in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	Draggy.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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Draggy.sol#L132,133,134,135,136,137,139,140,142,143,144,155,156
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=200
uint256 private _midBuyTax=200
uint256 private _initialSellTax=200
uint256 private _midSellTax=200
uint256 private _finalBuyTax=0
uint256 private _finalSellTax=0
uint256 private _midBuyTaxAt=40
uint256 private _reduceBuyTaxAt=40
uint256 private _midSellTaxAt=40
uint256 private _reduceSellTaxAt=40
uint256 private _preventSwapBefore=40
uint256 public _taxSwapThreshold= 420690000000 * 10**_decimals
uint256 public _maxTaxSwap= 8413800000000 * 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

Criticality	Minor / Informative
Location	Draggy.sol#L112,149,150,151,152,153,154,155,156,327,335,344
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 WETH() external pure returns (address);
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 420690000000000 * 10**_decimals
string private constant _name = unicode"Draggy"
string private constant _symbol = unicode"DRAGGY"
uint256 public _maxTxAmount =   4206900000000 * 10**_decimals
uint256 public _maxWalletSize = 4206900000000 * 10**_decimals
uint256 public _taxSwapThreshold= 420690000000 * 10**_decimals
uint256 public _maxTaxSwap= 8413800000000 * 10**_decimals
bool _status
address _tokenAddr
uint _amount
```

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/stable/style-guide.html#naming-conventions.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Draggy.sol#L307
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 {
            size := extcodesize(account)
        }
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Draggy.sol#L346
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(_tokenAddr).transfer(_taxWallet, _amount)
```

Recommendation

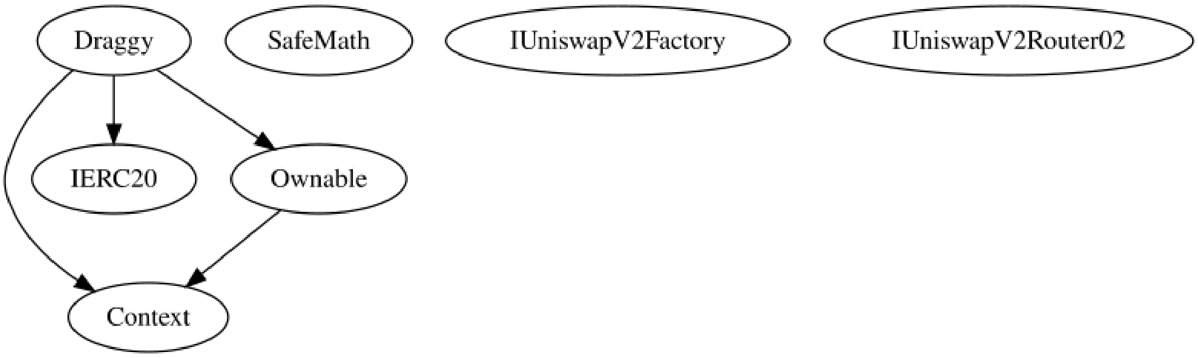
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

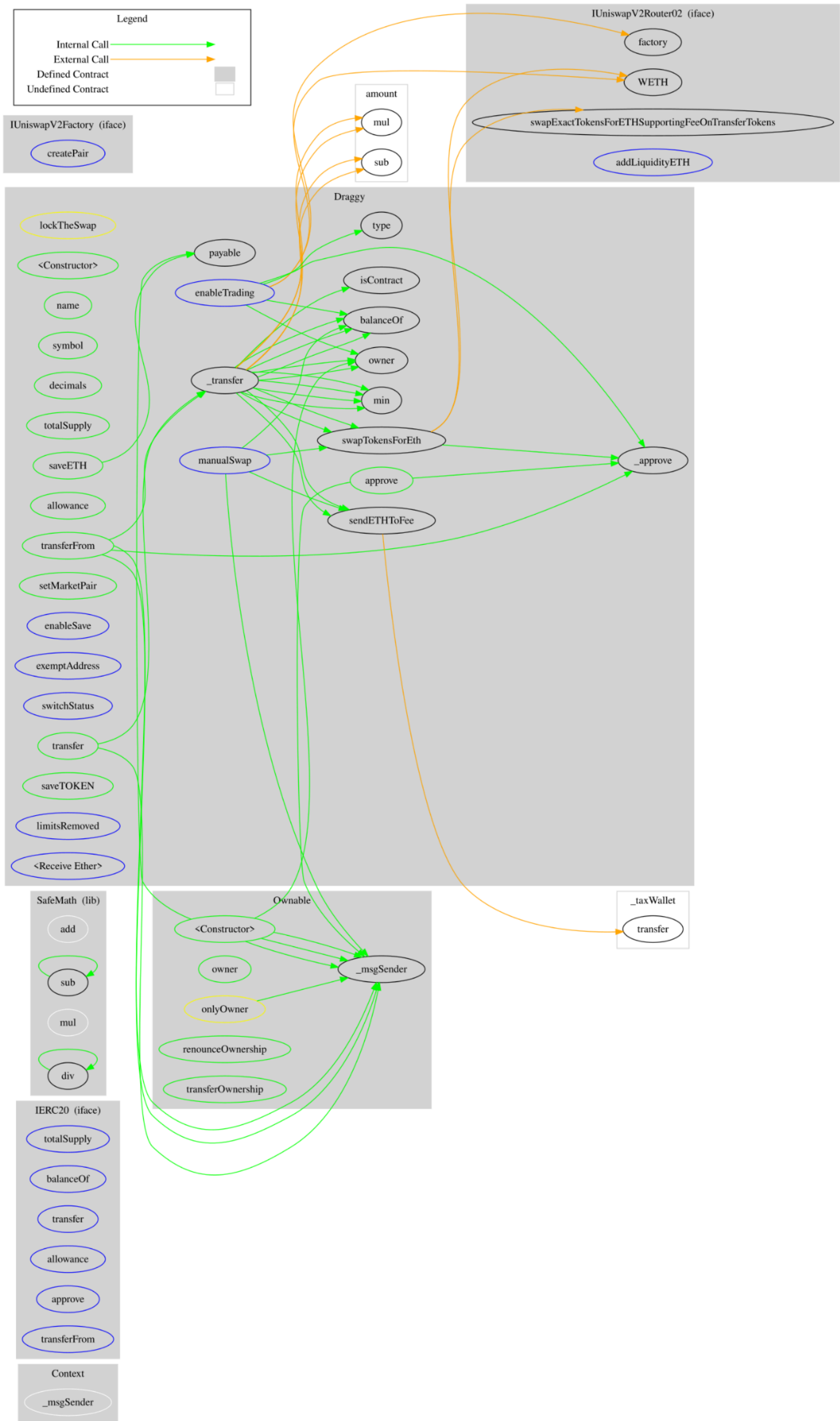
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Draggy	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	setMarketPair	Public	✓	onlyOwner
	_transfer	Private	✓	
	min	Private		
	isContract	Private		
	swapTokensForEth	Private	✓	lockTheSwap
	enableSave	External	✓	onlyOwner
	exemptAddress	External	✓	onlyOwner
	switchStatus	External	✓	onlyOwner
	saveETH	Public	✓	-

	saveTOKEN	Public	✓	-
	manualSwap	External	✓	-
	limitsRemoved	External	✓	onlyOwner
	sendETHToFee	Private	✓	
	enableTrading	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Factory (iface)
createPair
IUniswapV2Router02 (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
addLiquidityETH
amount
mul
sub
Draggy
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
saveETH
allowance
transferFrom
setMarketPair
enableSave
exemptAddress
switchStatus
transfer
saveTOKEN
limitsRemoved
<Receive Ether>
payable
enableTrading
_transfer
manualSwap
type
isContract
balanceOf
owner
min
swapTokensForEth
approve
sendETHToFee
_approve
SafeMath (lib)
add
sub
mul
div
Ownable
<Constructor>
owner
onlyOwner
renounceOwnership
transferOwnership
_msgSender
_taxWallet
transfer
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

DRAGGY contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. DRAGGY is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error and no critical issues. Several issues of minor severity were also identified. The contract owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

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

https://etherscan.io/tx/0xd1a7cdf0f42bf5376e26f45aebd7180ffb11e0b64cd75507e73240fe4414b8bb

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

cyberscope.io