



Cyberscope

Audit Report

LegacyX Token

June 2023

Network MATIC

Address 0x3Ad0af29E3826B8F22BC5DB10fFb42412ae67864

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
●	PAV	Pair Address Validation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	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
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Review	5
Audit Updates	5
Source Files	5
Findings Breakdown	6
PAV - Pair Address Validation	7
Description	7
Recommendation	7
PVC - Price Volatility Concern	8
Description	8
Recommendation	8
RSW - Redundant Storage Writes	9
Description	9
Recommendation	9
DDP - Decimal Division Precision	10
Description	10
Recommendation	10
OCTD - Transfers Contract's Tokens	11
Description	11
Recommendation	12
ULTW - Transfers Liquidity to Team Wallet	13
Description	13
Recommendation	13
RSML - Redundant SafeMath Library	14
Description	14
Recommendation	14
IDI - Immutable Declaration Improvement	15
Description	15
Recommendation	15
L02 - State Variables could be Declared Constant	16
Description	16
Recommendation	16
L04 - Conformance to Solidity Naming Conventions	17
Description	17
Recommendation	18
L07 - Missing Events Arithmetic	19
Description	19

Recommendation	19
L09 - Dead Code Elimination	20
Description	20
Recommendation	21
L13 - Divide before Multiply Operation	22
Description	22
Recommendation	22
L15 - Local Scope Variable Shadowing	23
Description	23
Recommendation	23
Functions Analysis	24
Inheritance Graph	31
Flow Graph	32
Summary	33
Disclaimer	34
About Cyberscope	35

Review

Contract Name	LEGX
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://polygonscan.com/address/0x3ad0af29e3826b8f22bc5db10ffb42412ae67864
Address	0x3ad0af29e3826b8f22bc5db10ffb42412ae67864
Network	MATIC
Symbol	LEGX
Decimals	18
Total Supply	1,777,777,777

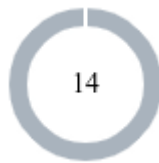
Audit Updates

Initial Audit	02 Jun 2023
---------------	-------------

Source Files

Filename	SHA256
LEGX.sol	67e5e71b3d4f2df640c364988bb78ae59b1927a72bbf698b1db39eb435313eaa

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

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	LEGX.sol#L609
Status	Unresolved

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
function setAutomatedMarketMakerPair(address pair, bool value) public  
onlyOwner {  
    require(pair != uniswapV3Pair, "The UniSwap pair cannot be removed  
from AutomatedMarketMakerPairs");  
    _setAutomatedMarketMakerPair(pair, value);  
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	LEGX.sol#L587
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function updateSwapTokensAtAmount(uint256 newAmount) external onlyOwner
returns (bool){
    swapTokensAtAmount = newAmount * (10**18);
    return true;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	LEGX.sol#L595,624,632,809
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 modifies the state of certain variables without checking if their current state is equal to the one given as an argument. As a result, the contract performs redundant storage writes.

```
_isExcludedFromFees[account] = excluded;  
marketingWallet = newMarketingWallet;  
expensesWallet = newWallet;  
buyBackEnabled = _buyBackEnabled;
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	LEGX.sol#L683,684,685,686,691,692,693,694,775,776,777,778
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
tokensForMarketing += fees * marketingSellFee / totalSellFees;  
tokensForExpenses += fees * expensesSellFee / totalSellFees;  
tokensForManagement += fees * managementSellFee / totalSellFees;  
tokensForDeveloper += fees * developerSellFee / totalSellF  
...
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	LEGX.sol#L801
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `recoverStuckTokens` function.

The `tokensForMarketing`, `tokensForExpenses`, `tokensForManagement` and `tokensForDeveloper` accumulate tokens that are meant to be swapped. If the owner calls the `recoverStuckTokens` function, then the contract will transfer all of its tokens to the owner. The contract's balance will become zero, but the contract does not reset the `tokensForMarketing`, `tokensForExpenses`, `tokensForManagement` and `tokensForDeveloper` variables back to zero. Afterwards, once the `swapBack` function gets triggered again, then these variables will have an amount greater than the contract's balance actual amount. As a result, there will be an inconsistency between the contract's balance and these variables.

```
function recoverStuckTokens(address tokenAddress, uint256 tokens) external onlyOwner
returns (bool success){
    return ERC20(tokenAddress).transfer(msg.sender, tokens);
}
```

Recommendation

The contract should reset the `tokensForMarketing` , `tokensForExpenses` , `tokensForManagement` and `tokensForDeveloper` values back to zero, when `recoverStuckTokens` is called and the contract transfers its balance to the owner. 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.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	LEGX.sol#L795
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the team wallet. These funds have been accumulated from fees collected from the contract. The owner may take advantage of it by calling the `recoverStuckETH` method.

```
function recoverStuckETH() external onlyOwner {
    (bool success,) = address(msg.sender).call{value:
address(this).balance}("Stuck ETH balance from contract address
recovered");
    require(success, "Failed. Either caller is not the owner or address is
not the contract address");
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped, since a huge amount may volatile the token's price. 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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	LEGX.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

Criticality	Minor / Informative
Location	LEGX.sol#L527,528,529,530,531,534,535,536,537,538,547,550,569,570
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.

```
marketingSellFee
expensesSellFee
managementSellFee
developerSellFee
totalSellFees
marketingBuyFee
expensesBuyFee
managementBuyFee
developerBuyFee
totalBuyFees
managementWallet
developerWallet
uniswapV3Router
uniswapV3Pair
```

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	LEGX.sol#L481,483
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.

```
bool public swapEnabled = true
uint256 public feeDivisor = 1000
```

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	LEGX.sol#L40,41,57,329,511,512,806
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);
event marketingWalletUpdated(address indexed newWallet, address indexed oldWallet);
event expensesWalletUpdated(address indexed newWallet, address indexed oldWallet);
uint256 _percentForMarketing
bool _buyBackEnabled
```

Recommendation

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

Find more information on the Solidity documentation

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

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	LEGX.sol#L588,808
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.

```
swapTokensAtAmount = newAmount * (10**18)
percentForMarketing = _percentForMarketing
```

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	LEGX.sol#L743
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 addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {  
    _approve(address(this), address(uniswapV3Router), tokenAmount);  
  
    uniswapV3Router.addLiquidityETH{value: ethAmount}(  
        ...  
        0,  
        0,  
        address(0xdead),  
        block.timestamp  
    );  
}
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	LEGX.sol#L683,684,685,686,690,691,692,693,694,775,785
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 loss of precision.

```
fees = amount.mul(totalBuyFees).div(feeDivisor)
tokensForDeveloper += fees * developerBuyFee / totalBuyFees
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	LEGX.sol#L521
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint256 totalSupply = 1777777777 * (10**18)
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IUniswapV3Pair	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	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV3Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Meta data		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-

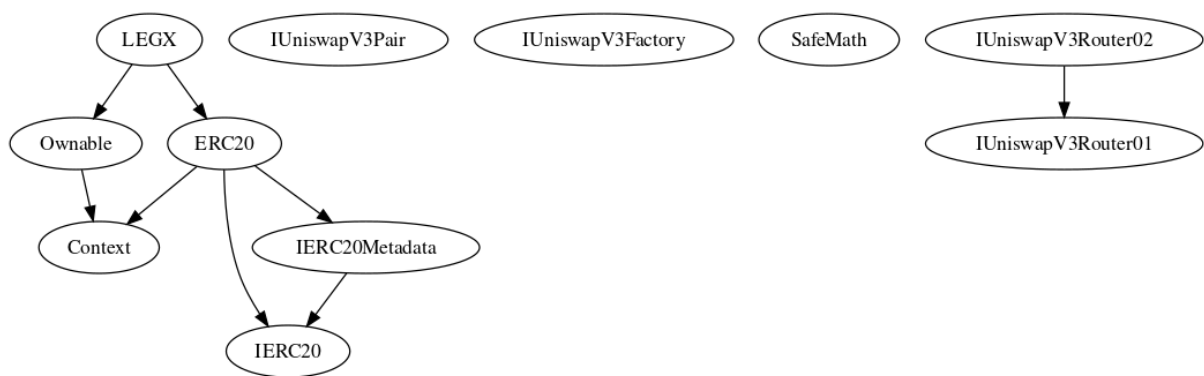
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_createInitialSupply	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-

	renounceOwnership	External	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV3Router01	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		-

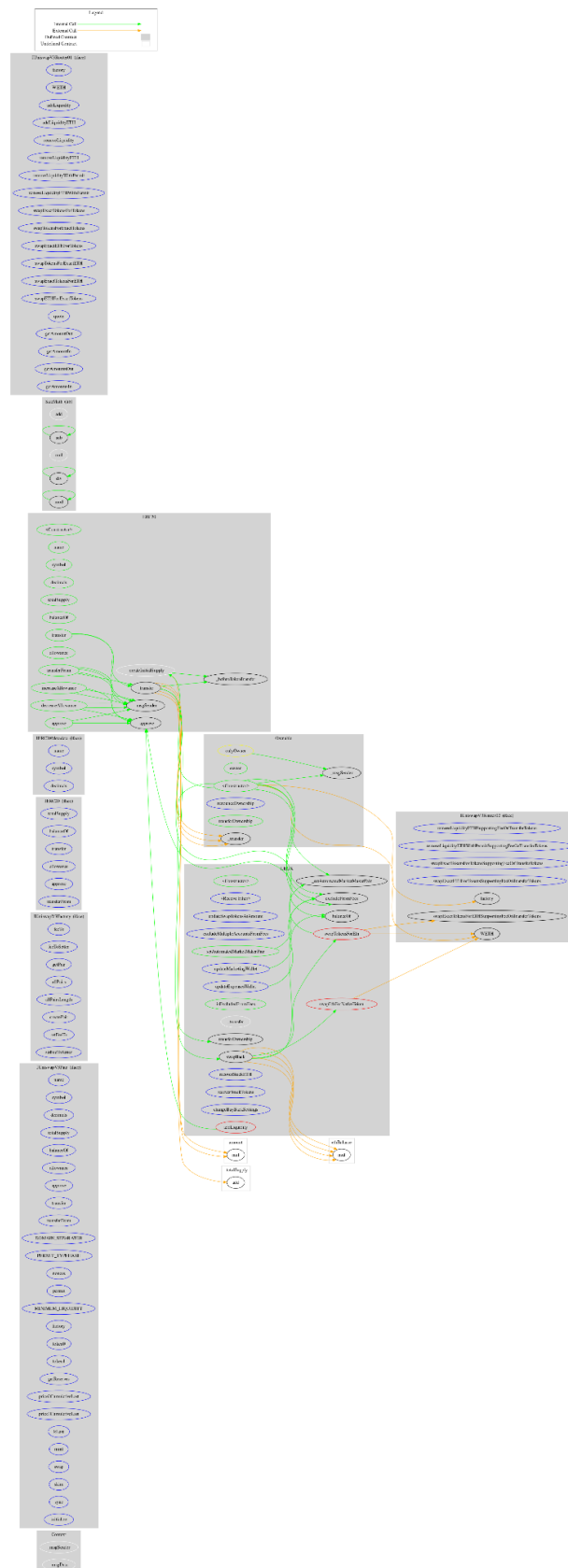
IUniswapV3Router02	Interface	IUniswapV3Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
LEGX	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	updateSwapTokensAtAmount	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	updateExpensesWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapEthForNativeToken	Private	✓	
	swapTokensForEth	Private	✓	

	addLiquidity	Private	✓	
	swapBack	Private	✓	
	recoverStuckETH	External	✓	onlyOwner
	recoverStuckTokens	External	✓	onlyOwner
	changeBuyBackSettings	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

LegacyX Token contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. LegacyX Token is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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 7% buy and sell 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>