Cyberscope

Audit Report

Lambro

June 2023

Network	BSC
Address	0x201AF44d9DfA5464F20B8dD8aA96Fc016d26E7C0
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
●	MLF	Misleading Liquidity Functionality	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	CR	Code Repetition	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	LambroToken
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x201af44d9dfa5464f20b8dd8aa96fc016d26e7c0
Address	0x201af44d9dfa5464f20b8dd8aa96fc016d26e7c0
Network	BSC
Symbol	LAMBRO
Decimals	18
Total Supply	69,420,000,000

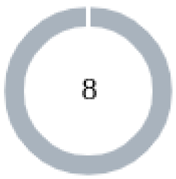
Audit Updates

Initial Audit	26 Jun 2023

Source Files

Filename	SHA256
LambroToken.sol	c1f14c3a8af6d0dbf0ec1844a024eadf06bbb804c477fc971d808b6df767f4cc

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	8

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

MLF - Misleading Liquidity Functionality

Criticality	Minor / Informative
Location	LambroToken.sol#L492
Status	Unresolved

Description

The contract implements a liquidity functionality, indicating that it intends to interact with liquidity pools. However, there is no implementation or function present that enables users or the contract itself to add liquidity to the desired pool. Additionally, the balance of the `lpAddress` address is used to receive the tokens from the liquidity pool fee (`lpFee`).

```
address public lpAddress ;

_balances[lpAddress] = _balances[lpAddress].add(lpFee);
```

Recommendation

The contract should revisit the liquidity functionality since sending tokens to the liquidity address does not have any effect.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	LambroToken.sol#L556,560
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 updates the `isExcludedFromFee` mapping even if the current state of an account is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function ExcludedFromFee(address account, bool) public onlyOwner {
    isExcludedFromFee[account] = true;
}

function IncludeInFee(address account, bool) public onlyOwner {
    isExcludedFromFee[account] = false;
}
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	LambroToken.sol#L483,507
Status	Unresolved

Description

The `buyCollectFee` and `SellbuyCollectFee` functions in the contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
function buyCollectFee(address account, uint256 amount/*, uint256
rate*/) private returns (uint256) {

    uint256 transferAmount = amount;
    uint256 charityFee = amount.mul(_charity).div(10000);
    uint256 lpFee = amount.mul(_lpFee).div(10000);

    if (lpFee > 0){
        transferAmount = transferAmount.sub(lpFee);
        _balances[lpAddress] = _balances[lpAddress].add(lpFee);
        _lpFeeTotal = _lpFeeTotal.add(lpFee);
        emit Transfer(account,lpAddress,lpFee);
    }

    if(charityFee > 0){
        transferAmount = transferAmount.sub(charityFee);
        _balances[charityAddress] =
_balances[charityAddress].add(charityFee);
        _charityTotal = _charityTotal.add(charityFee);
        emit Transfer(account,charityAddress,charityFee);
    }
    return transferAmount;
}
```

```
   function SellbuyCollectFee(address account, uint256 amount) private
returns (uint256) {

        uint256 transferAmount = amount;
        uint256 charityFee = amount.mul(_charity).div(10000);
        uint256 burningFee = amount.mul(_burningFee).div(10000);
        uint256 lpFee = amount.mul(_lpFee).div(10000);

...

    if (lpFee > 0){
        transferAmount = transferAmount.sub(lpFee);
        _balances[lpAddress] = _balances[lpAddress].add(lpFee);
        _lpFeeTotal = _lpFeeTotal.add(lpFee);
        emit Transfer(account,lpAddress,lpFee);
    }
    if(charityFee > 0){
        transferAmount = transferAmount.sub(charityFee);
        _balances[charityAddress] =
_balances[charityAddress].add(charityFee);
        _charityTotal = _charityTotal.add(charityFee);
        emit Transfer(account,charityAddress,charityFee);
    }

    return transferAmount;
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	LambroToken.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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	LambroToken.sol#L363,364,365,372,373,374,375,376,381,382,384
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.

```
string private _name = "Lambro"
string private _symbol = "LAMBRO"
uint8 private _decimals = 18
uint256 internal _totalSupply = 69420000000 *10**18
uint256 public _charity = 100
uint256 public _burningFee = 100
uint256 public _lpFee = 100
uint256 public _inbetweenFee_ = 300
address public charityAddress  =
0x2CD2BD245b2C563C98356B543d6C9a1aBf7353C5
address public burningAddress =
0x0000000000000000000000000000000000000000
address public inbetweenAddress =

0x80Dc1B16eB733Ca49B289f6B39090EC0dC0222a8
```

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	LambroToken.sol#L368,369,372,373,374,375,376,377,378,379,380,507,556,560,564
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.

```
mapping(address => uint256) internal _balances
mapping(address => mapping(address => uint256)) internal _allowances
uint256 internal _totalSupply = 69420000000 *10**18
uint256 public _charity = 100
uint256 public _burningFee = 100
uint256 public _lpFee = 100
uint256 public _inbetweenFee_ = 300
uint256 public _charityTotal
uint256 public _burningFeeTotal
uint256 public _lpFeeTotal
uint256 public _inbetweenFeeTotal

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	LambroToken.sol#L4,234,261,343
Status	Unresolved

Description

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

```
pragma solidity 0.8.19;
pragma solidity ^0.8.0;
```

Recommendation

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

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

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	LambroToken.sol#L4,234,261
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

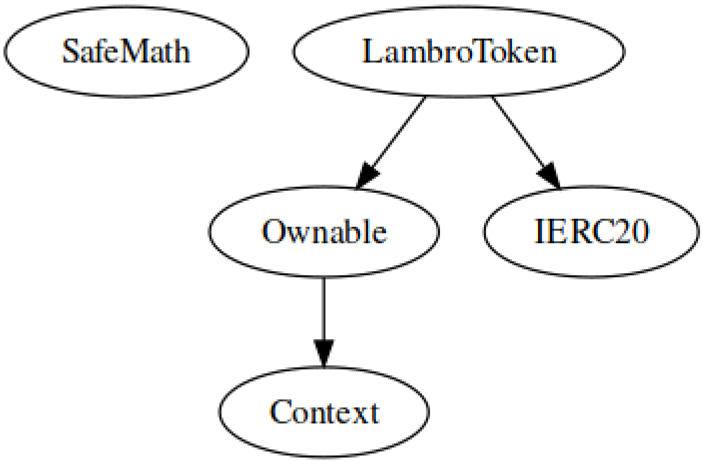
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

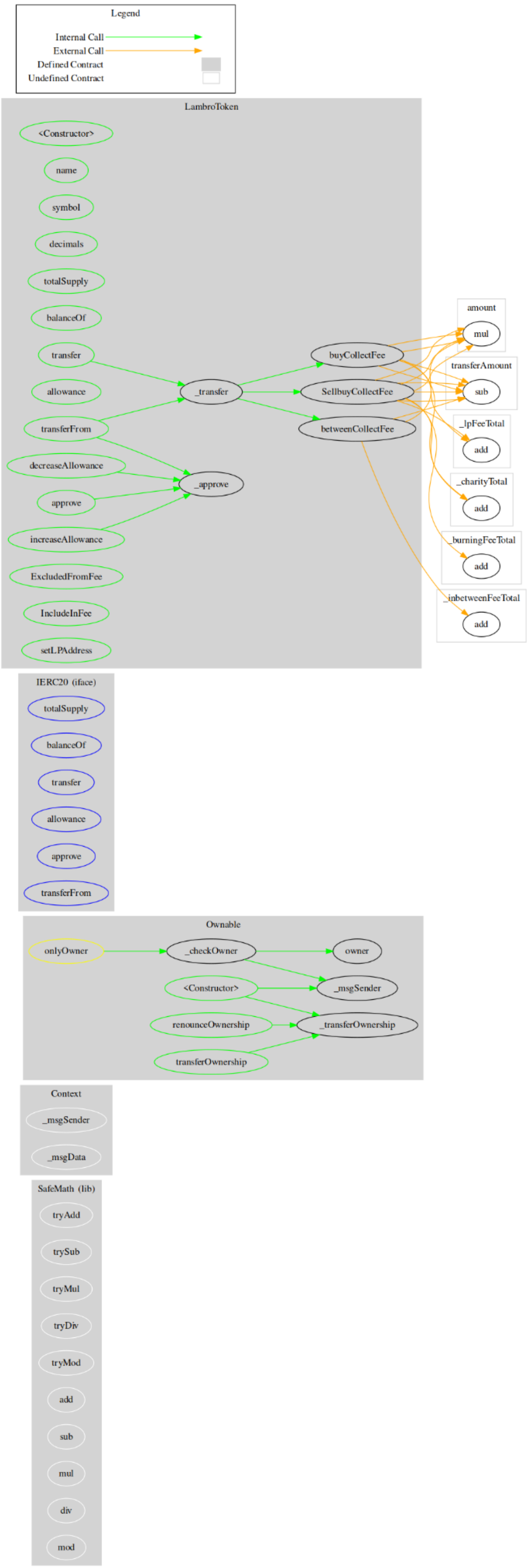
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
LambroToken	Implementation	IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-

	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	buyCollectFee	Private	✓	
	SellbuyCollectFee	Private	✓	
	betweenCollectFee	Private	✓	
	ExcludedFromFee	Public	✓	onlyOwner
	IncludeInFee	Public	✓	onlyOwner
	setLPAddress	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
LambroToken
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
decreaseAllowance
approve
increaseAllowance
ExcludedFromFee
IncludeInFee
setLPAddress
_transfer
_approve
buyCollectFee
SellbuyCollectFee
betweenCollectFee
amount
mul
transferAmount
sub
_lpFeeTotal
add
_charityTotal
add
_burningFeeTotal
add
_inbetweenFeeTotal
add
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
Context
_msgSender
_msgData
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod

Summary

Lambro contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Lambro 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 3% 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