Cyberscope

Audit Report

Libra

February 2023

SHA256 5031430cc2613c32736d598037d3075985a2a09e61592a013dbd09a5bc2041b8

Audited by © cyberscope

Table of Contents

Review

Contract Name	Libra
Compiler Version	v0.8.11+commit.d7f03943
Optimization	200 runs
Testing Deploy	https://testnet.bscscan.com/address/0xf413ff91768bfc0542c3b6af377295a1b325f555
Address	0xf413ff91768bfc0542c3b6af377295a1b325f555
Network	BSC_TESTNET
Symbol	LBR
Decimals	18
Total Supply	500,000

Audit Updates

Initial Audit	22 Feb 2023 https://github.com/cyberscope-io/audits/tree/main/lbr/v1/audit.pdf
Corrected Phase 2	24 Feb 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/token/ERC20/ERC20.sol	5031430cc2613c32736d598037d307598 5a2a09e61592a013dbd09a5bc2041b8
@openzeppelin/contracts/token/ERC20/extension s/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb982616 6689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.so l	94f23e4af51a18c2269b355b8c7cf4db80 03d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a 23a4baa0b5bd9add9fb6d6a1549814a
contracts/Libra.sol	a7538741dd25d898f4f177f7eb261a4df5 85153c634ceaa33c741b85f9cc9d45

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L19	Stable Compiler Version	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/LibraToken.sol#L20,27
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 wallet address that receives the fees from the contract's transactions is used directly at the `_transfer()` function. One of the primary issues with using a hardcoded wallet address is that if the address changes, the contract will need to be updated and redeployed. This can be problematic in situations where the contract is already live on the blockchain, and users are interacting with it.

Additionally, using a hardcoded wallet address can make the contract more difficult to maintain and update. For example, if there are multiple instances in the code where the same address is used, updating the address in all instances can be a tedious and error-prone process.

```
_transfer(msg.sender, 0xdb2052dE1B1788f37E61340A2A2773bD4559C04d, taxAmount);
```

Recommendation

The team is advised to define variables for commonly used addresses, so they can be easily updated if needed.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/LibraToken.sol#L9,11
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.

```
uint256 private constant _taxRate = 7
uint256 private constant _totalSupply = 500000 * DECIMALS
```

Recommendation

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

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/LibraToken.sol#L3
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 ^0.8.11;
```

Recommendation

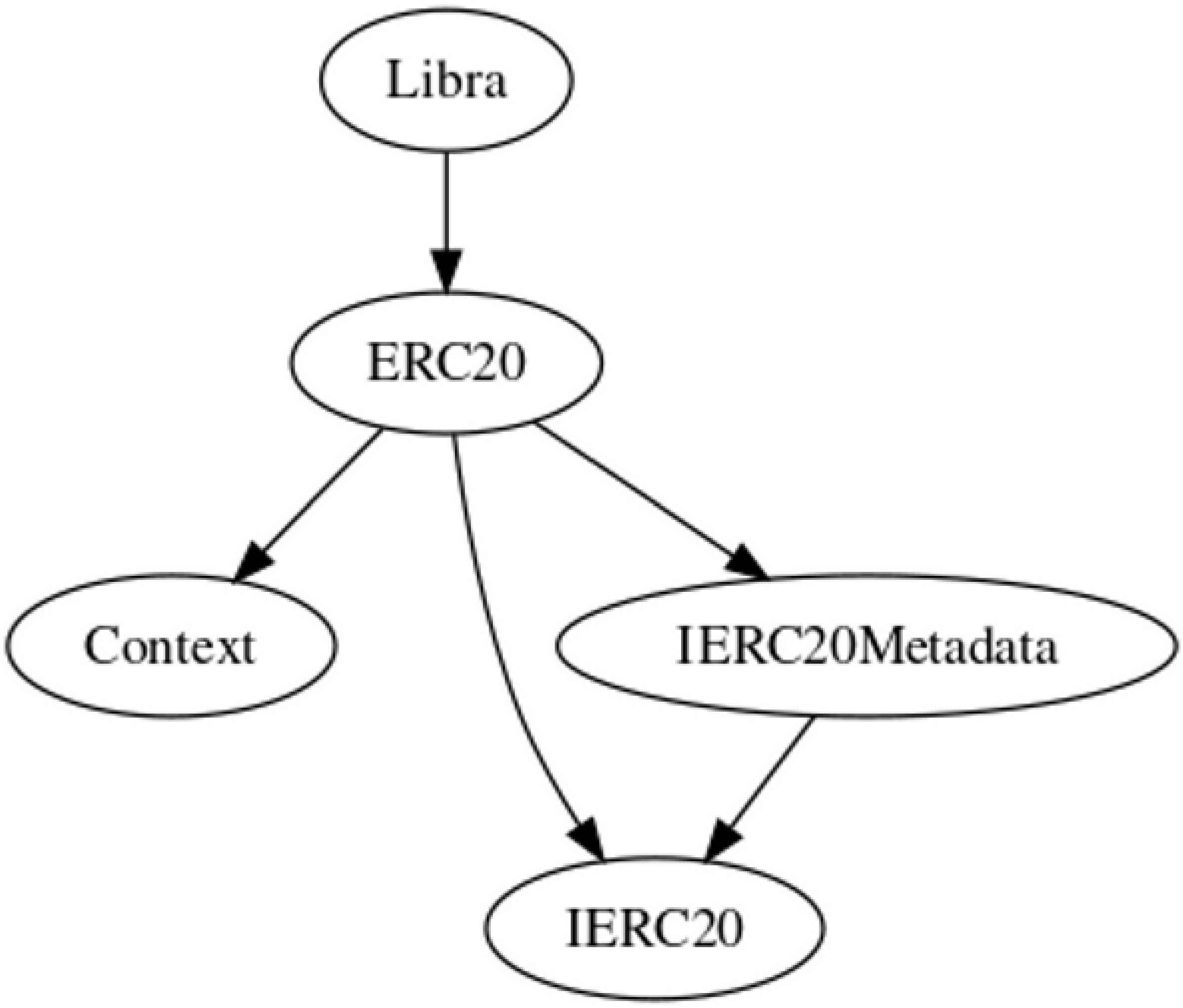
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

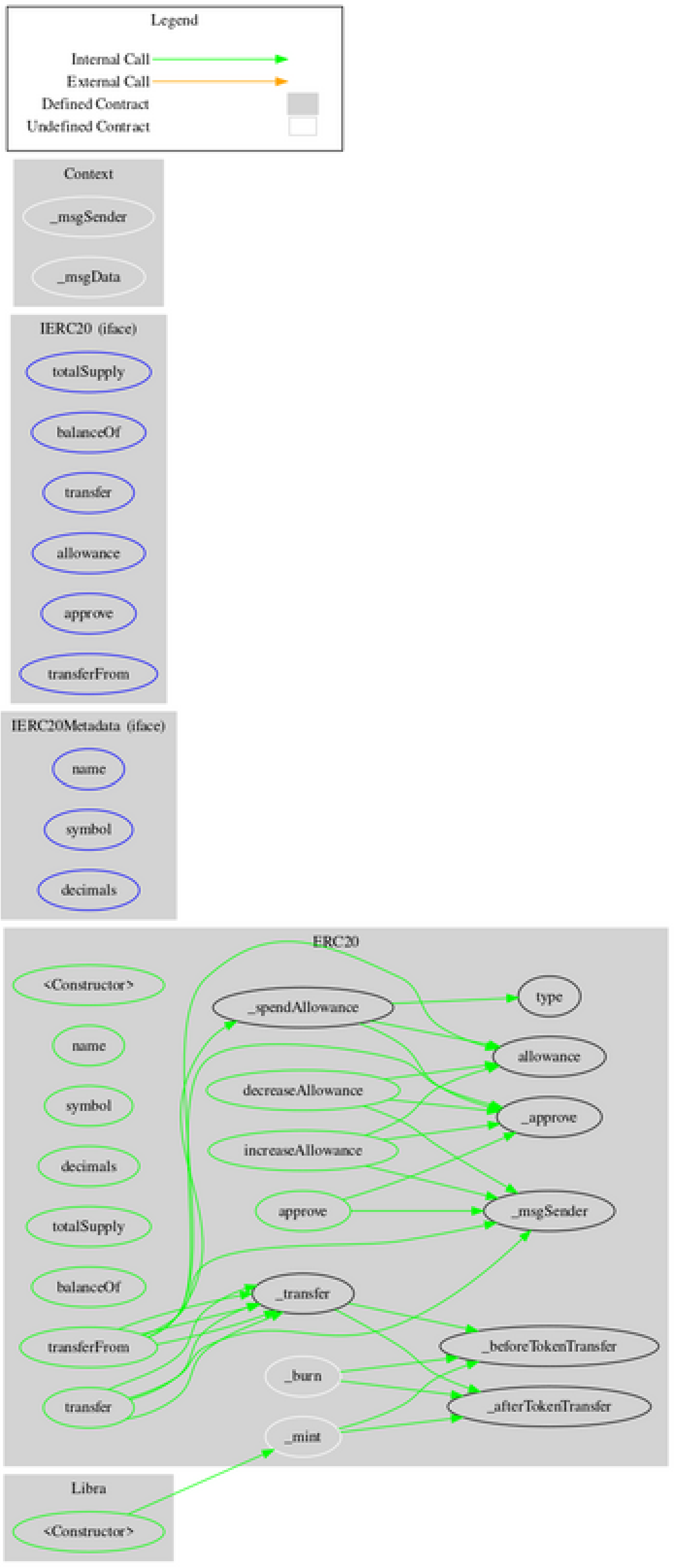
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IERC20Metadata	Interface	IERC20		

	name	External		-
	symbol	External		-
	decimals	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Libra	Implementation	ERC20		
		Public	✓	ERC20
	transfer	Public	✓	-
	transferFrom	Public	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Context
_msgSender
_msgData
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
IERC20Metadata (iface)
name
symbol
decimals
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
decreaseAllowance
increaseAllowance
approve
_transfer
_burn
_mint
type
allowance
_approve
_msgSender
_beforeTokenTransfer
_afterTokenTransfer
Libra
<Constructor>

Summary

Libra 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% fees.

Disclaimer

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