Cyberscope

Audit Report

MoonLabs Token

March 2023

SHA256 aaa9a39a5f923d2ddb524d1957cd9963b400587d20c6a2156f11589bc087fdb0

Audited by © cyberscope

Table of Contents

Review

Contract Name	MoonLabs
Testing Deploy	https://testnet.bscscan.com/address/0xb7574d2e1fa1cff6cd6e930b18a4c2363d9e9162
Symbol	MLAB
Decimals	18
Total Supply	0

Audit Updates

Initial Audit	24 Mar 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC20/ERC20.sol	bce14c3fd3b1a668529e375f6b70ffdf9cef8c4e410ae99608be5964d98fa701
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/token/ERC721/IERC721.sol	c7703068bac02fe1cdf109e38faf10399c66eb411e4c9ae0d70c009eca4bf5ef
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol	29c75e69ce173ff8b498584700fef76bc81498c1d98120e2877a1439f0c31b5a
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38
contracts/MoonLabs.sol	aaa9a39a5f923d2ddb524d1957cd9963b400587d20c6a2156f11589bc087fdb0

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	Passed

ST - Stops Transactions

Criticality	Critical
Location	MoonLabs.sol#L190
Status	Unresolved

Description

The contract stops the sales for all users including the owner. A statement in the `_transfer` function is incorrect, which prevents any further transactions from being processed. This can lead to significant disruption in the application's functionality and prevent users from completing important actions, such as buying or selling tokens.

```
uint16[] memory indexArray;
address[] memory addressArray;
```

Recommendation

It is recommended to update incorrect code statements to ensure that transactions can be processed as intended. Additionally, we recommend thoroughly testing the updated code to verify that it functions as expected and does not introduce any new issues.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RAV	Reentrancy Attack Vulnerability	Unresolved
●	RCS	Redundant Code Statement	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	ZD	Zero Division	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MMSI	Max Mint Supply Inconsistency	Unresolved
●	MEE	Misleading Event Emission	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved

RAV - Reentrancy Attack Vulnerability

Criticality	Critical
Location	MoonLabs.sol#L201
Status	Unresolved

Description

The contract is vulnerable to a reentrancy attack, which can occur if a buyer initiates a trade using a contract address as the seller or by initiating a crypto trade from a contract. For instance,

- A user tries to transfer tokens to enter the nft reward mechanism to receive as many rewards as he wants.
- A reentrance attack will occur if the `nftOwner` implements a receive callback, they will have the ability to execute any method again within the same execution thread.

```
if (nftBalance >= nftPayout) {
    address nftOwner = nftContract.ownerOf(nftIndex);
    /// Send eth to index holder
    (bool sent, ) = payable(nftOwner).call{ value: nftPayout }("");
    ...
}
```

Recommendation

The contract could disallow the use of contract addresses for transfer transactions, or alternatively, the contract may employ a ReentrancyGuard to prevent reentrancy issues.

RCS - Redundant Code Statement

Criticality	Minor / Informative
Location	MoonLabs.sol#L192
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 is utilizing a redundant boolean variable. The functionality of `distributed` variable could be replaced with a pre-existing variable.

```
bool distributed;

for (uint i = 0; i < maxNftDistribution; i++) {
    /// Set distributed to true if not true
    if (!distributed) distributed = true;
    ...
}

/// Emit event if nft payout
if (distributed) emit DistributeNftPayout(addressArray, indexArray,
nftPayout);
```

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.

The contract could remove the distributed variable and use `maxNftDistribution` variable. For instance,

```
if (maxNftDistribution > 0) emit DistributeNftPayout(addressArray,
indexArray, nftPayout);
```

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	MoonLabs.sol#L201
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The contract could transfer `nftPayout` to zero addresses.

```
(bool sent, ) = payable(nftOwner).call{ value: nftPayout }("");
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

- The variable `nftOwner` should not be set to zero address.

ZD - Zero Division

Criticality	Minor / Informative
Location	MoonLabs.sol#L298
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

The variable `totalTokenTax` could be set to zero.

```
function _swapAndDistribute() private lockTheSwap {
    uint8 totalTokenTax = buyTax.totalTax + sellTax.totalTax;
    uint8 burnTax = buyTax.burnTax + sellTax.burnTax;
    uint8 liquidityTax = buyTax.liquidityTax + sellTax.liquidityTax;

    uint liquidityTokenCut = ((swapThreshold * liquidityTax) /
totalTokenTax) /
    ...
}
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow execution of the corresponding statements.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/MoonLabs.sol#L169
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` 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 setTokensToSellForTax(uint _swapThreshold) external onlyOwner
{
swapThreshold = _swapThreshold;
}
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
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.

The total Ethereum fees and the liquidity cut will not be splitted as expected.

```
function _swapAndDistribute() private lockTheSwap {
    uint8 totalTokenTax = buyTax.totalTax + sellTax.totalTax;
    uint8 burnTax = buyTax.burnTax + sellTax.burnTax;
    uint8 liquidityTax = buyTax.liquidityTax + sellTax.liquidityTax;

    uint liquidityTokenCut = ((swapThreshold * liquidityTax) /
totalTokenTax) / 2;
    uint burnTokenCut;

    /// If burns are enabled
    if (buyTax.burnTax != 0 || sellTax.burnTax != 0) {
      burnTokenCut = (swapThreshold * burnTax) / totalTokenTax;
      /// Send tokens to dead address
      super._transfer(address(this), address(0xdead), burnTokenCut);
    }

    _swapTokens(swapThreshold - liquidityTokenCut - burnTokenCut);

    uint ethBalance = address(this).balance;

    uint totalEthFee = (totalTokenTax - (liquidityTax / 2) - burnTax);

    /// Distribute to team and treasury
    (treasuryWallet).call{ value: (ethBalance * (buyTax.treasuryTax +
sellTax.treasuryTax)) / totalEthFee }("");
    (teamWallet).call{ value: (ethBalance * (buyTax.teamTax +
sellTax.teamTax)) / totalEthFee }("");

    /// Add ETH to nft balance
    nftBalance += (ethBalance * buyTax.nftTax + sellTax.nftTax) /
totalEthFee;

    /// Add tokens to liquidity
    _addLiquidity((liquidityTokenCut), ((ethBalance * liquidityTax) /
totalEthFee) / 2);
  }
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

MMSI - Max Mint Supply Inconsistency

Criticality	Minor / Informative
Location	contracts/MoonLabs.sol#L212
Status	Unresolved

Description

The contract uses a fixed value for the maximum mint amount instead of retrieving it from the linked NFT contract. This approach can lead to a discrepancy between the actual available max mint supply of NFTs and the preset fixed max mint supply.

```
if (nftIndex < 500) {
    nftIndex++;
} else {
    nftIndex = 1;
}
```

Recommendation

It is recommended to retrieve the maximum mint value from the linked NFT contract. This approach will ensure that the maximum mint value is updated according to the available supply of NFTs. Retrieving the maximum mint value from the NFT contract will also ensure that the contract's behavior is consistent with the linked NFT contract. The contract could initial the max mint amount on the contract constructor.

MEE - Misleading Event Emission

Criticality	Minor / Informative
Location	contracts/MoonLabs.sol#L224
Status	Unresolved

Description

The contract is emitting misleading events. These event messages do not accurately reflect contract flow, making it difficult to track blockchain events.

The contract emits a `DistributeNftPayout` even if no payouts are sent successfully.

```
/// Emit event if nft payout
if (distributed) emit DistributeNftPayout(addressArray, indexArray,
nftPayout);
```

Recommendation

The team is advised to carefully review the source code in order to address these issues. To accelerate the debugging process and mitigate these issues, the team should use accurate event emission.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MoonLabs.sol#L32,33,34,35
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 = "Moon Labs"
string private _symbol = "MLAB"
uint8 private _decimals = 9
uint private _supply = 100000000
```

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	MoonLabs.sol#L120,124,130,136,141,146,151,156,196
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.

```
uint _nftPayout
uint8 _maxNftDistribution
address payable _treasuryWallet
address payable _teamWallet
address payable _liqWallet
address _address
bool _taxSwap
uint _swapThreshold
```

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	MoonLabs.sol#L121,126,197
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.

```
nftPayout = _nftPayout
maxNftDistribution = _maxNftDistribution
swapThreshold = _swapThreshold
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MoonLabs.sol#L220,221,222,260,305
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.

```
uint16[] memory indexArray
address[] memory addressArray
bool distributed
uint fees
uint burnTokenCut
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MoonLabs.sol#L70,71,72
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
treasuryWallet = _treasuryWallet
teamWallet = _teamWallet
liqWallet = _liqWallet
```

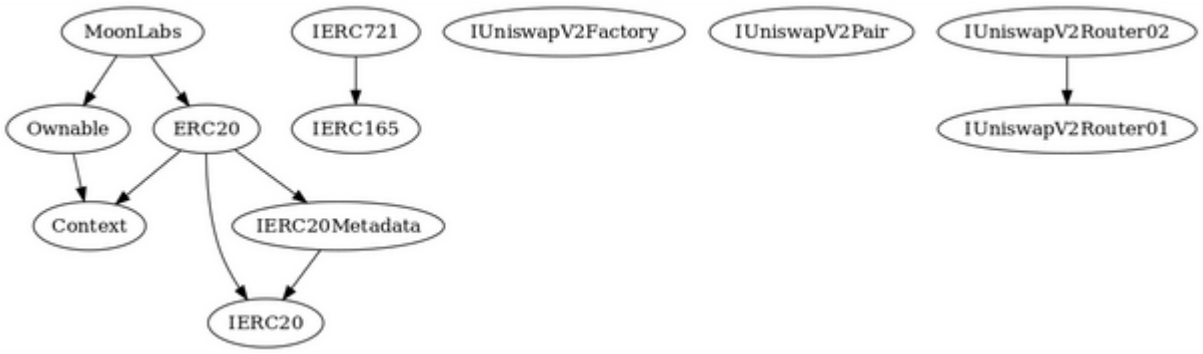
Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

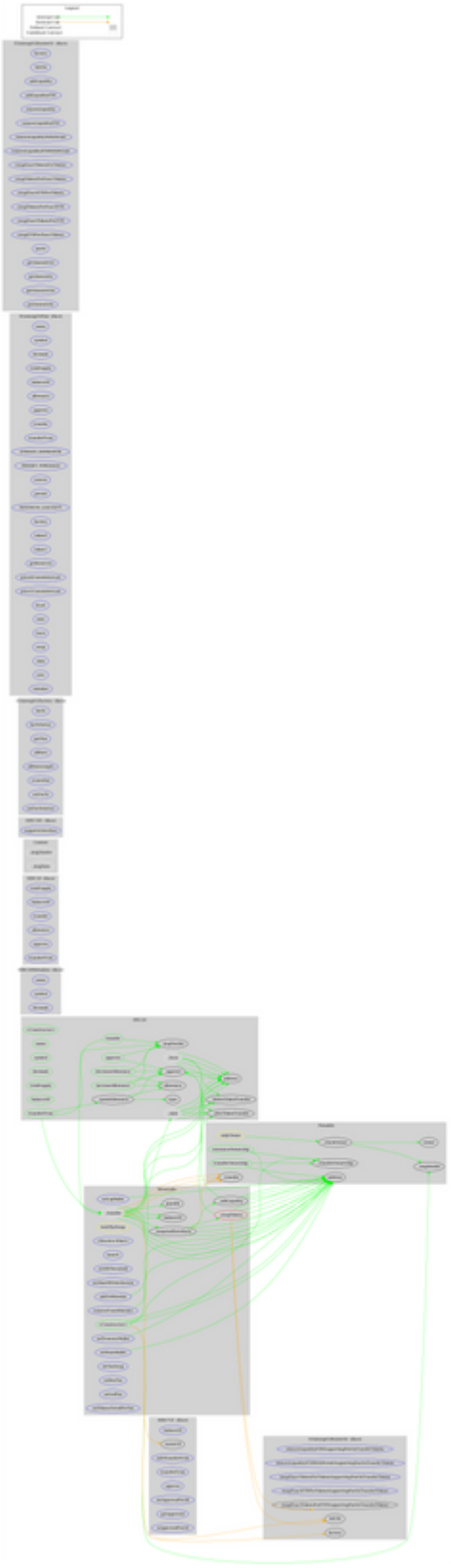
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MoonLabs	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	launch	External	✓	onlyOwner
	setNftThreshold	External	✓	onlyOwner
	setMaxNftDistribution	External	✓	onlyOwner
	setTreasuryWallet	External	✓	onlyOwner
	setTeamWallet	External	✓	onlyOwner
	setLiqWallet	External	✓	onlyOwner
	addToWhitelist	External	✓	onlyOwner
	removeFromWhitelist	External	✓	onlyOwner
	setTaxSwap	External	✓	onlyOwner
	setBuyTax	External	✓	onlyOwner
	setSellTax	External	✓	onlyOwner
	setTokensToSellForTax	External	✓	onlyOwner
	_transfer	Internal	✓	
	_swapTokens	Private	✓	lockTheSwap
	_swapAndDistribute	Private	✓	lockTheSwap
	_addLiquidity	Private	✓	lockTheSwap

Inheritance Graph



Flow Graph



Summary

Moonlabs contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that do not operate as expected. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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