Cyberscope

Audit Report

Moonlabs

March 2023

SHA256 ef21aa22c5ec7b6a4c710a60b57d007f6d872145b5ae7afc485ae023922e5c02
5e585113d852b7be27774d6338591cb425e004f0b13bb25a3fc851869fd9a13a
fdd25ae863e8b3369e07a3dbd88488e3508b3068cc6f3c3cd42ce5f6d8744141
edfb7bf95df1d4e8aae8369e1e335e3fa040dd0a6367e8497537dbcfce776b82
f3ef41718d2bd2a2fd45234063921d24a71bafafa0387eeab0b6b1c41484311b

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 4
Testing Deploy 5
Findings Breakdown 6
Introduction 7
MoonLabsWhitelist 8
Roles 8
MoonLabsReferral 9
Roles 9
MoonLabsVesting 10
Roles 10
MoonLabsTokenLocker 11
Roles 11
MoonLabsLiquidityLocker 12
Roles 12
Diagnostics 13
PTM - Pair Token Mocking 14
Description 14
Recommendation 14
TPP - Token Pair Prevalidation 15
Description 15
Recommendation 15
AAO - Accumulated Amount Overflow 16
Description 16
Recommendation 16
PTAI - Potential Transfer Amount Inconsistency 17
Description 17
Recommendation 18
MSC - Missing Sanity Checks 19
Description 19
Recommendation 19
CO - Code Optimization 20
Description 20
Recommendation 20
L04 - Conformance to Solidity Naming Conventions 21
Description 21

Recommendation 22
L07 - Missing Events Arithmetic 23
Description 23
Recommendation 23
L14 - Uninitialized Variables in Local Scope 24
Description 24
Recommendation 24
L16 - Validate Variable Setters 25
Description 25
Recommendation 25
Functions Analysis 26
Inheritance Graph 36
Flow Graph 37
Summary 38
Disclaimer 39
About Cyberscope 40

Review

Audit Updates

Initial Audit	09 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v1/audit.pdf
Corrected Phase 2	20 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v2/audit.pdf
Corrected Phase 3	24 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v3/audit.pdf
Corrected Phase 4	04 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v4/audit.pdf
Corrected Phase 5	20 Apr 2023

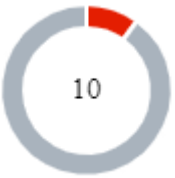
Source Files

Filename	SHA256
IDEXRouter.sol	f6af1340f54f9d239f19282c0177481296b1b10699e094523136f381714afb30
MoonLabsLiquidityLocker.sol	edfb7bf95df1d4e8aae8369e1e335e3fa040dd0a6367e8497537dbcfce776b82
MoonLabsTokenLocker.sol	fdd25ae863e8b3369e07a3dbd88488e3508b3068cc6f3c3cd42ce5f6d8744141
MoonLabsVesting.sol	5e585113d852b7be27774d6338591cb425e004f0b13bb25a3fc851869fd9a13a
MoonLabsWhitelist.sol	ef21aa22c5ec7b6a4c710a60b57d007f6d872145b5ae7afc485ae023922e5c02

Testing Deploy

Contract Name	Explorer
MoonLabsLiquidityLocker	https://testnet.bscscan.com/address/0xD239Ac752cD84ea566dF1cB56F6356c8ef966204
MoonLabsReferral	https://testnet.snowtrace.io/address/0xdCF65098B0873153A86C2408dFD012c4847fF299
MoonLabsTokenLocker	https://testnet.bscscan.com/address/0x7d12fCaF71Ebdd5D03Ca6c2a98ffd0A20c6577D4
MoonLabsVesting	https://testnet.bscscan.com/address/0x37BF4bA8FA330Bb9E8e22ec909b65D9090d16089
MoonLabsWhitelist	https://testnet.bscscan.com/address/0x9B38A1d4B70Ac8C2faF05528306a3048149341d4

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

Introduction

The Moonlab ecosystem consists of five contracts as upgradable proxies. Two utility contracts and three locker contracts.

MoonLabsWhitelist

The MoonLabsWhitelist contract is used for creating whitelists for Moon Labs products. Whitelisting a token allows users to waive all fees on related Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` is responsible for:

- Adding or removing a whitelisted address.
- Claim all contract deposited balance.
- Claim all usdContract token balance.
- The users have the authority to:
- Purchase a whitelist with or without a discount code.
- Check the whitelisted address.
- Add or remove the blacklisted pairs.
- Configure contract address feeCollector, ReferralContract, Router, USDContract, MlabToken
- Set Mlab discount percent.

MoonLabsReferral

The MoonLabsReferral smart contract is used for creating and managing referral codes. It allows users to create referral codes for customers to use while purchasing Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Delete referral code.
- Add reserved codes.
- Assign or remove reserved codes.
- Add or remove addMoonLabsContract.
- Claim all contract deposited balance.
- The user has the authority to:
- Check active codes.
- Get the address to the referral code.
- Get referral code to address.
- Add rewards earned for a code.
- Create, delete, transfer, and reserve referral codes.

MoonLabsVesting

The MoonLabsVesting Contract allows token owners to create ERC20 token locks. The Contract supports creating multiple vesting instances, choosing between linear and standard Locks, and transferring Locks to other addresses. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function. Lock creators cannot modify locks once they have been created, and withdraw owners cannot extend or change lock details.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Create a lock MLAB token.
- Get MLAB fee.

MoonLabsTokenLocker

The MoonLabsTokenLocker contract is designed to allow users to create locks for ERC20 tokens. Lock creators can extend, transfer, add to, and split locks, but cannot unlock tokens prematurely. Users can create lock instances for the same token and choose either a linear or standard lock. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like important, address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.
- Get claimable tokens.
- Create lock MLAB tokens.
- Relock MLAB tokens.
- split lock MLAB tokens.
- Get MLAB fee.

MoonLabsLiquidityLocker

The MoonLabsLiquidityLocker contract is responsible for creating liquidity locks for Uniswap-based AMM tokens. The main purpose of the contract is to allow users to create locks for selected wallets with the option to choose between standard or linear lock types. The lock type is determined by the start date, with the default being a standard lock. The locked tokens remain locked until their respective unlock date without any exceptions, and lock owners are not allowed to unlock them prematurely. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like importance, address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PTM	Pair Token Mocking	Unresolved
●	TPP	Token Pair Prevalidation	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MSC	Missing Sanity Checks	Unresolved
●	CO	Code Optimization	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

PTM - Pair Token Mocking

Criticality	Critical
Location	MoonLabsWhitelist.sol#L321
Status	Unresolved

Description

The contract does not properly validate token pairs, leaving it susceptible to potential token mocking attacks. Attackers could create fake tokens that mimic legitimate ones, potentially leading to unauthorized transactions and loss of funds for the contract and its users.

```
function getIsWhitelisted(
    address _address,
    bool pair
) public view override returns (bool) {
    if (tokenToWhitelist[_address]) return true;
    /// Return false if not pair or pool
    if (!pair) return false;
    /// Check for v2 pairs
    if (_checkV2Pair(_address)) return true;
    /// Check for v3 pools
    if (_checkV3Pool(_address)) return true;
    return false;
}
```

Recommendation

It is recommended to implement robust validation checks to ensure that only legitimate token pairs are used in the contract. For Instance, to ensure that the pair address is legitimate, the contract could leverage a trusted address to access the factory contract's `getPair` function and validate that the returned pair address is identical to the provided argument address.

TPP - Token Pair Prevalidation

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol#L1020MoonLabsVesting.sol#L822MoonLabsTokenLocker.sol#L1197
Status	Unresolved

Description

The contract allows users to initiate swap transactions between two tokens without first checking if a token pair exists. This could result in the loss of funds if the contract is unable to find a liquidity pool for the specified token pair.

```
function handleBurns() private {
    /// Check if the threshold is met
    uint _burnMeter = burnMeter;
    if (burnMeter >= burnThreshold) {
      /// Buy tokenToBurn via Uniswap router and send to the
dead address
      address[] memory path = new address[](2);
      path[0] = routerContract.WETH();
      path[1] = address(tokenToBurn);

routerContract.swapExactETHForTokensSupportingFeeOnTransferToke
ns{ value: _burnMeter }(0, path,
0x000000000000000000000000000000000000dEaD, block.timestamp);
      _burnMeter = 0;
      burnMeter = _burnMeter;
    }
}
```

Recommendation

It is recommended to pre-validate that a token pair exists before allowing users to initiate swap transactions. A valid pair address should have token0, token1, factory

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.solMoonLabsTokenLocker.solMoonLabsVesting.sol
Status	Unresolved

Description

The contract is using the variable `nonce` to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint64 public nonce; /// Unique lock identifier
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol MoonLabsLiquidityLocker.sol#L923 MoonLabsVesting.sol#L606 MoonLabsTokenLocker.sol#L808
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
usdContract.safeTransferFrom(
    address(this),
    msg.sender,
    usdContract.balanceOf(address(this))
);

IERC20Upgradeable(tokenAddress).safeTransferFrom(
    from,
    address(this),
    amount
);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

MSC - Missing Sanity Checks

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol MoonLabsVesting.sol MoonLabsTokenLocker.sol
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.

```
function initialize(...){...}
function splitLockETH(address to, uint64 _nonce, uint amount )
external payable {...}
function splitLockPercent(address to, uint64 _nonce, uint
amount) external {...}
function splitLockPercent(address to, address
withdrawalAddress, uint64 _nonce, uint amount ) external {...}
```

Recommendation

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

- The `initializer` addresses shouldn't be set to zero addresses.
- The `to` addresses shouldn't be set to zero addresses.
- The `withdrawalAddress` addresses shouldn't be set to zero addresses.

CO - Code Optimization

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L641
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 `depositAmount` and `totalDeposit` are the same. Hence the MathUpgradeable.mulDiv calculation is redundant.

```
MathUpgradeable.mulDiv(
    amountSent,
    depositAmount,
    totalDeposited
),
```

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.

- Redundant code statements could be removed.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L44,45,49,83,96,116,145,155,165,175,185,193,202,211,220,229,238,247,257,282,286,297 MoonLabsVesting.sol#L47,48,322,358,359,404,413,422,431,439,447,456,466,476,485,518,557 MoonLabsTokenLocker.sol#L47,48,351,386,387,388,426,463,487,517,553,583,623,671,680,689,698,706,714,722,730,739,749,759,768,778,789,834,878 MoonLabsLiquidityLocker.sol#L63,64,316,346,376,402,424,452,485,514,550,595,604,613,622,630,638,646,654,663,673,683,692,702,713,747,785
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.

```
address _mlabToken
address _feeCollector
uint _costUSD
address _address
uint8 _codeDiscount
uint8 _codeCommission
address _referralAddress
address _routerAddress
address _usdAddress
uint8 _mlabDiscountPercent
uint64 _nonce
uint _burnThreshold
uint _ethLockPrice
uint8 _burnPercent

...
```

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	MoonLabsWhitelist.sol#L186,195,213,260 MoonLabsVesting.sol#L432,469,478 MoonLabsTokenLocker.sol#L699,752,761 MoonLabsLiquidityLocker.sol#L623,676,685
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.

```
costUSD = _costUSD
codeDiscount = _codeDiscount
codeCommission = _codeCommission
mlabDiscountPercent = _mlabDiscountPercent
burnThreshold = _burnThreshold
burnPercent = _burnPercent
 setRelockPrice(uint _ethReloc

t");
        percentLockPrice = _percentLo

onlyOwner {
        requir
```

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	MoonLabsWhitelist.sol#L390,395,416,421
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.

```
address _token0
address _token1
```

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	MoonLabsWhitelist.sol#L52 MoonLabsVesting.sol#L55 MoonLabsTokenLocker.sol#L55 MoonLabsLiquidityLocker.sol#L71
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.

```
feeCollector = _feeCollector
c ethSplitPrice; /// Price i
```

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**
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsLiquidityLocker	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockMLAB	External	Payable	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-

	nukeLock	External	✓	-
	relockMLAB	External	Payable	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockMLAB	External	Payable	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setMlabToken	External	✓	onlyOwner
	setMlabDiscountPercent	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-

	getNonceFromTokenAddress	External		-
	getLock	External		-
	getMLABFee	Public		-
	getClaimableTokens	Public		-
	_buyWithMLAB	Private	✓	
	_relock	Private	✓	
	_splitLock	Private	✓	
	_createLock	Private	✓	
	_transferAndCalculate	Private	✓	
	_transferAndCalculateWithFee	Private	✓	
	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	
	_handleBurns	Private	✓	
	_distributeCommission	Private	✓	nonReentrant
	_deleteLockInstance	Private	✓	
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			

	getIsWhitelisted	External		-
MoonLabsTokenLocker	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockMLAB	External	✓	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	setLockWithdrawalAddress	Public	✓	-
	relockMLAB	External	Payable	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockMLAB	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner

	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setMlabToken	External	✓	onlyOwner
	setMlabDiscountPercent	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	getMLABFee	Public		-
	getClaimableTokens	Public		-
	_buyWithMLAB	Private	✓	
	_relock	Private	✓	
	_splitLock	Private	✓	
	_createLocks	Private	✓	
	_calculateTotalDeposited	Private		
	_transferAndCalculate	Private	✓	

	_transferAndCalculateWithFee	Private	✓	
	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	
	_handleBurns	Private	✓	
	_distributeCommission	Private	✓	nonReentrant
	_deleteLockInstance	Private	✓	
	_calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsVesting	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockMLAB	External	✓	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-

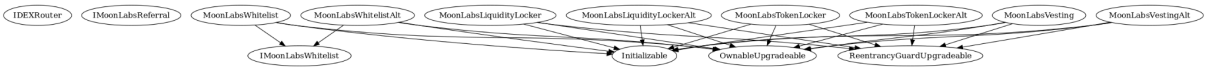
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferVestingOwnership	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setMlabToken	External	✓	onlyOwner
	setMlabDiscountPercent	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getInstance	External		-
	getMLABFee	Public		-
	getClaimableTokens	Public		-
	_buyWithMLAB	Private	✓	
	_createLocks	Private	✓	
	_calculateTotalDeposited	Private		
	_transferAndCalculate	Private	✓	

	_transferAndCalculateWithFee	Private	✓	
	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	
	_deleteVestingInstance	Private	✓	
	_distributeCommission	Private	✓	nonReentrant
	_handleBurns	Private	✓	
	_calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarnedUSD	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsWhitelist	Implementation	Initializable, IMoonLabsWhitelist, OwnableUpgradeable		
	initialize	Public	✓	initializer
	purchaseWhitelistMLAB	External	✓	-
	purchaseWhitelist	External	✓	-
	purchaseWhitelistWithCode	External	✓	-

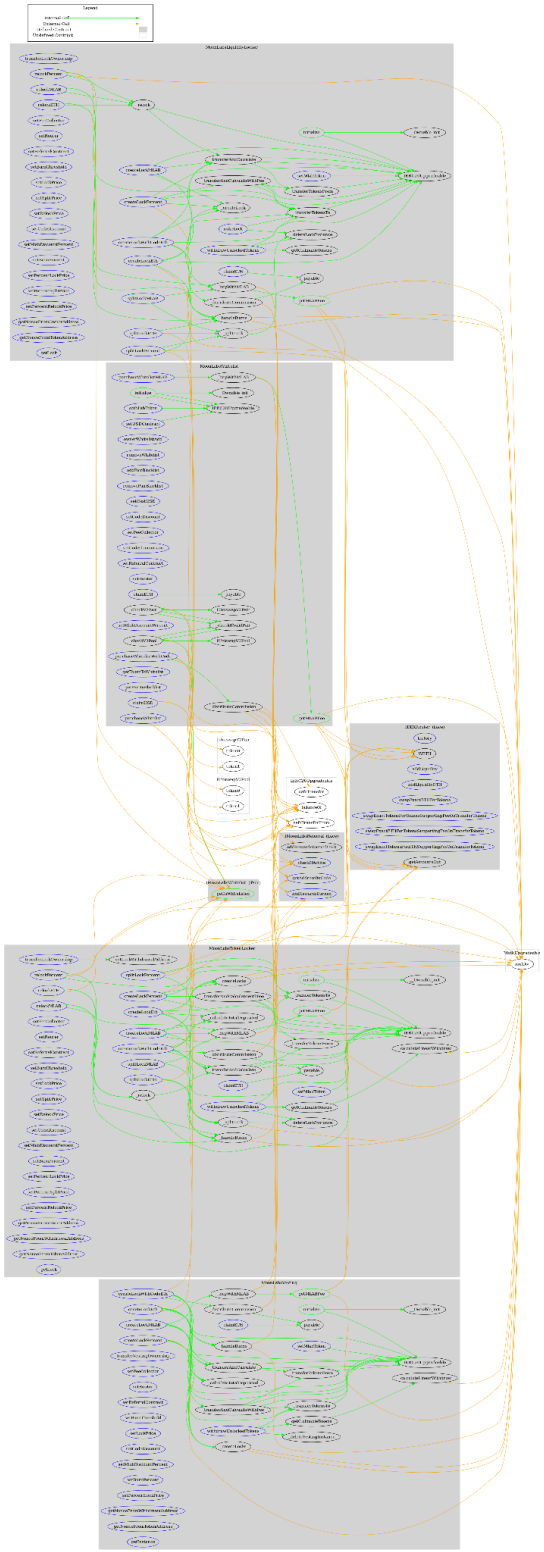
	ownerWhitelistAdd	External	✓	onlyOwner
	removeWhitelist	External	✓	onlyOwner
	addPairBlacklist	External	✓	onlyOwner
	removePairBlacklist	External	✓	onlyOwner
	setCostUSD	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setUSDContract	External	✓	onlyOwner
	setMlabToken	External	✓	onlyOwner
	setMlabDiscountPercent	External	✓	onlyOwner
	claimETH	External	✓	onlyOwner
	claimUSD	External	✓	onlyOwner
	getTokenToWhitelist	External		-
	getPairToBlacklist	External		-
	getIsWhitelisted	Public		-
	getMLABFee	Public		-
	_buyWithMLAB	Private	✓	
	_distributeCommission	Private	✓	
	_checkV2Pair	Private		
	_checkV3Pool	Private		

	_checkIfValidPair	Private		

Inheritance Graph



Flow Graph



Summary

Moonlab's contracts implement a utility, financial, and locker mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

MoonLabsStakingCards

April 2023

Network	ETH
Address	0xE6FBD188fFc0604E7Aa3290C303C9Dc11ec98D53
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Findings Breakdown 3
Introduction 4
Roles 4
Diagnostics 5
MT - Mints Tokens 6
Description 6
Recommendation 6
L02 - State Variables could be Declared Constant 7
Description 7
Recommendation 7
L04 - Conformance to Solidity Naming Conventions 8
Description 8
Recommendation 8
Functions Analysis 10
Inheritance Graph 17
Flow Graph 18
Summary 19
Disclaimer 20
About Cyberscope 21

Review

Contract Name	MoonLabsStakingCards
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	3500 runs
Explorer	https://etherscan.io/address/0xe6fbd188ffc0604e7aa3290c303c9dc11ec98d53
Address	0xe6fbd188ffc0604e7aa3290c303c9dc11ec98d53
Network	ETH
Symbol	MLSC

Audit Updates

Initial Audit	24 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabsd/v1/nft.pdf
Corrected Phase 2	06 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabsd/v2/nft.pdf
Corrected Phase 3	20 Apr 2023

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	3

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	3	0	0	0

Introduction

The MoonLabsStakingCards contract extends ERC721Enumerable and Ownable contracts from the OpenZeppelin library. It allows for the creation and management of non-fungible tokens (NFTs). The contract has a maximum supply of 500 NFTs, which can be minted by the contract owner only. The contract also provides a function for getting all the token IDs owned by a particular address, setting the base token URI, and claiming all ETH held in the contract. The tokenURI function returns the compiled token URI for a given token ID.

Roles

The contract consists of the owner role. The owner is responsible for the base URI configuration and for minting tokens.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	MT	Mints Tokens	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

MT - Mints Tokens

Criticality	Minor / Informative
Status	Unresolved

Description

The contract owner has the authority to mint tokens.

```
function ownerMint(address _to, uint amount) external onlyOwner
{
    uint supply = totalSupply();
    require(supply + amount <= MAX_SUPPLY, "Max supply
reached");
    for (uint i = 1; i <= amount; i++) {
      _safeMint(_to, supply + i);
    }
}
```

Recommendation

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/MoonLabsStakingCards.sol#L29
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 public BASE_EXTENSION = ".json";
```

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	contracts/MoonLabsStakingCards.sol#L29,39,54,87
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.

```
string public BASE_EXTENSION = ".json";
function getTokenIds(address _owner) external view returns
(uint[] memory)
function ownerMint(address _to, uint amount) external onlyOwner
uint _id

...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-

	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_ownerOf	Internal		
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_requireMinted	Internal		
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	

	__unsafe_increaseBalance	Internal	✓	
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	
	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
IERC721Metadata	Interface	IERC721		
	name	External		-
	symbol	External		-

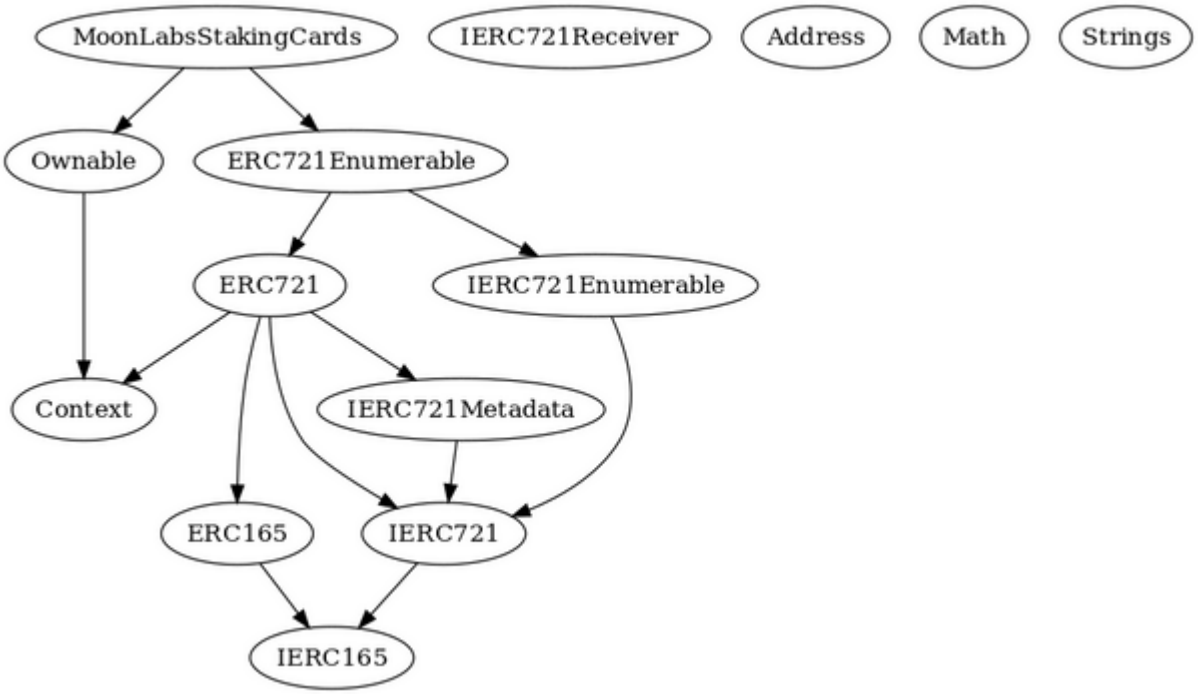
	tokenURI	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	

	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
IERC165	Interface			
	supportsInterface	External		-
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		

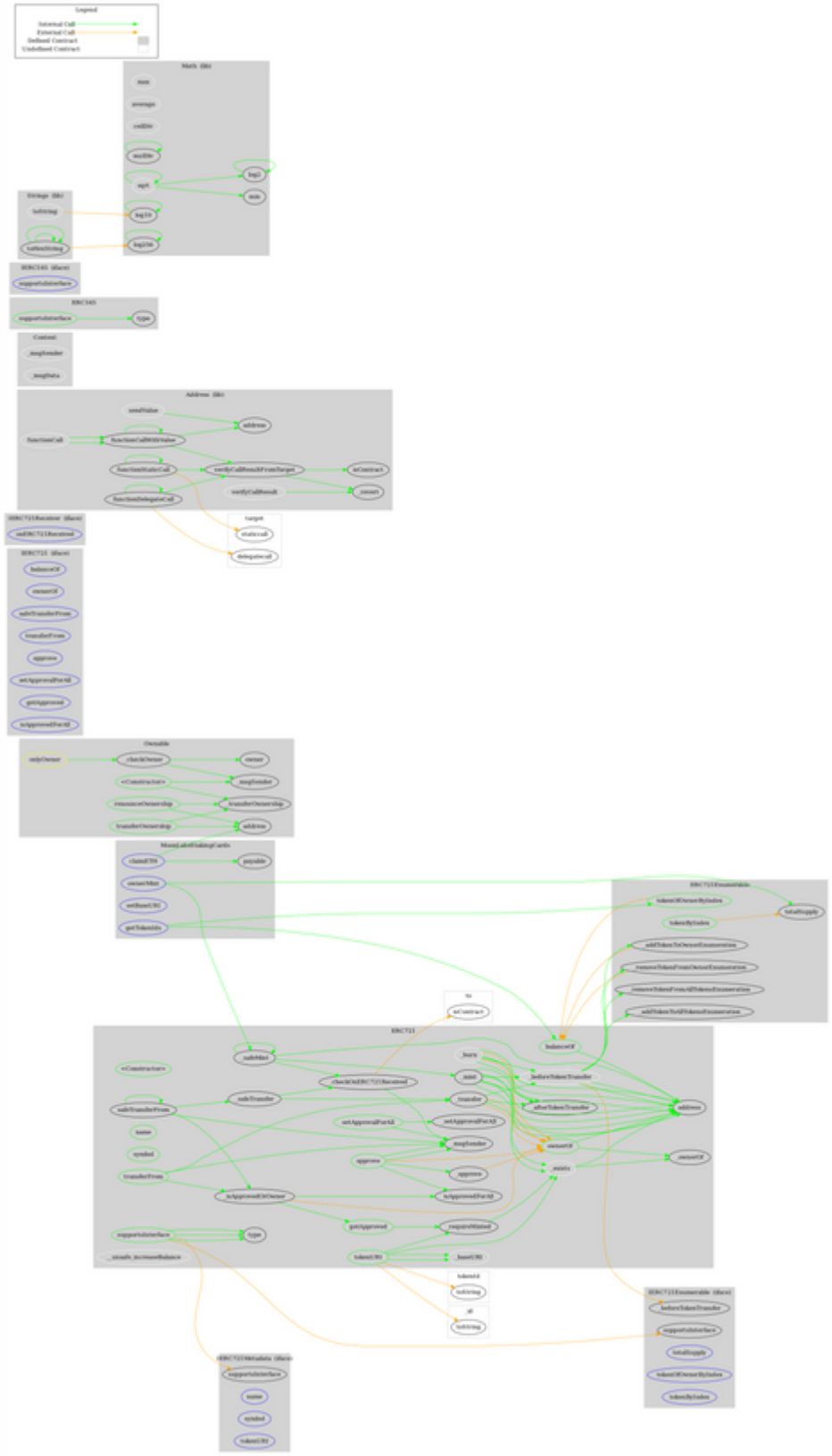
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
MoonLabsStakingCards	Implementation	ERC721Enumerable, Ownable		
		Public	✓	ERC721
	getTokenIds	External		-
	ownerMint	External	✓	onlyOwner
	setBaseURI	External	✓	onlyOwner

	claimETH	External	Payable	onlyOwner
	tokenURI	Public		-
	_baseURI	Internal		

Inheritance Graph



Flow Graph



Summary

MoonLabsStakingCards contract implements an nft mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

MoonLabs Token

April 2023

Network	ETH
Address	0x70B951F7d4Bb28BCE55Ee5374C95349324846833
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 3
Analysis 4
Diagnostics 5
RCS - Redundant Code Statement 6
Description 6
Recommendation 6
MSC - Missing Sanity Check 7
Description 7
Recommendation 7
DDP - Decimal Division Precision 8
Description 8
Recommendation 9
MMSI - Max Mint Supply Inconsistency 10
Description 10
Recommendation 10
L04 - Conformance to Solidity Naming Conventions 11
Description 11
Recommendation 11
L07 - Missing Events Arithmetic 12
Description 12
Recommendation 12
L16 - Validate Variable Setters 13
Description 13
Recommendation 13
Functions Analysis 14
Inheritance Graph 16
Flow Graph 17
Summary 18
Disclaimer 19
About Cyberscope 20

Review

Contract Name	MoonLabs
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	3500 runs
Explorer	https://etherscan.io/address/0x70b951f7d4bb28bce55ee5374c95349324846833
Address	0x70b951f7d4bb28bce55ee5374c95349324846833
Network	ETH
Symbol	MLAB
Decimals	9
Total Supply	100.000.000

Audit Updates

Initial Audit	24 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabsd/v1/token.pdf
Corrected Phase 2	06 Apr 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabsd/v2/token.pdf
Corrected Phase 3	20 Apr 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249a a4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC20/ERC20.sol	b39a02b7d35d9837f6f008bcdfe826c566c 1da17aa12fcbac12ebe79c6b1a9af
@openzeppelin/contracts/token/ERC20/extensions /IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166 689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db800 3d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/token/ERC721/IERC721. sol	c7703068bac02fe1cdf109e38faf10399c66 eb411e4c9ae0d70c009eca4bf5ef
@openzeppelin/contracts/utils/Address.sol	8160a4242e8a7d487d940814e5279d934e 81f0436689132a4e73394bab084a6d
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a2 3a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/IERC 165.sol	701e025d13ec6be09ae892eb029cd83b30 64325801d73654847a5fb11c58b1e5
@uniswap/v2-core/contracts/interfaces/IUniswapV 2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e19 0cc982db5771ffeef8d8d1f962a0d
@uniswap/v2-periphery/contracts/interfaces/IUnis wapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d6 1679947d158cba4ee0a1da60cf663
@uniswap/v2-periphery/contracts/interfaces/IUnis wapV2Router02.sol	a2900701961cb0b6152fc073856b972564f 7c798797a4a044e83d2ab8f0e8d38
contracts/MoonLabs.sol	a4699ae8c5b8a8c1421f92246b745085aa 8636548b570b6f8ad3ed0746fa3cc1

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
●	RCS	Redundant Code Statement	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MMSI	Max Mint Supply Inconsistency	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved

RCS - Redundant Code Statement

Criticality	Minor / Informative
Location	MoonLabs.sol#L255
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 rewardsSent = false;

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

/// Emit event if nft payout
if (rewardsSent) 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#L269
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 be zero address.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	MoonLabs.sol#L349
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 taxes might not be splitted as expected.

```
uint addToLiquidityHalf = ((swapThreshold * liquidityTax) /
    totalTokenTax) / 2;

_swapTokens(swapThreshold - addToLiquidityHalf - burnTokenCut);

uint ethBalance = address(this).balance - nftBalance;

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

/// Distribute to team and treasury
if (buyTax.treasuryTax + sellTax.treasuryTax > 0) {
    (treasuryWallet).call{
        value: (ethBalance *
            (buyTax.treasuryTax + sellTax.treasuryTax)) /
totalSellFee
    }("");
}

if (buyTax.teamTax + sellTax.teamTax > 0) {
    (teamWallet).call{
        value: (ethBalance * (buyTax.teamTax +
sellTax.teamTax)) /
            totalSellFee
    }("");
}

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

/// Add tokens to liquidity
if (addToLiquidityHalf > 0) {
    _addLiquidity(
        (addToLiquidityHalf),
        ((ethBalance * liquidityTax) / totalSellFee) / 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	MoonLabs.sol#L260
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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MoonLabs.sol#L129,133,139,145,150,155,160,165,205
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#L130,135,207
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MoonLabs.sol#L79,80,81
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

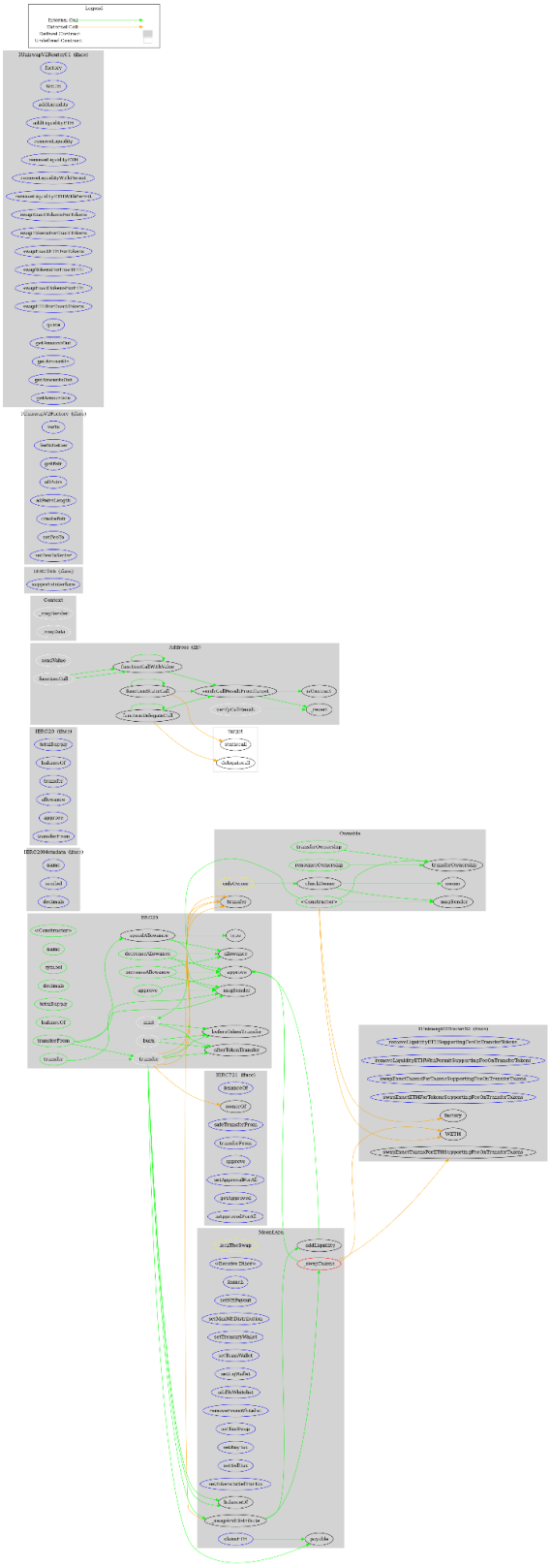
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MoonLabs	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	launch	External	✓	onlyOwner
	setNftPayout	External	✓	onlyOwner
	setMaxNftDistribution	External	✓	onlyOwner
	setTreasuryWallet	External	✓	onlyOwner
	setTeamWallet	External	✓	onlyOwner
	setLiqWallet	External	✓	onlyOwner
	addToWhitelist	External	✓	onlyOwner
	removeFromWhitelist	External	✓	onlyOwner
	setTaxSwap	External	✓	onlyOwner
	setBuyTax	External	✓	onlyOwner
	setSellTax	External	✓	onlyOwner
	setTokensToSellForTax	External	✓	onlyOwner
	claimETH	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. Moonlabs 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.

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

Cyberscope

Audit Report

Moonlabs

March 2023

SHA256 c021ea3c57bfbba72ee8097b0f50c50b75041ef3ec8fcecbcedca640b783a490
f3ef41718d2bd2a2fd45234063921d24a71bafafa0387eeab0b6b1c41484311b
feb646c4c147c0c44e0115a3055e12d9f1491e9d7d86d47923c23fe1a0bece73
b2cea2f09dfd04c62fea3649a435324f71aa9920ca9ff4dc72d6df531733fad8
7cef1147938ea185aa39c1c9807e8e8373896f8cf290901bf7f6ef8c1032bde1

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 4
Testing Deploy 4
Audit Updates 4
Source Files 5
Introduction 6
MoonLabsWhitelist 6
Roles 6
MoonLabsReferral 7
Roles 7
MoonLabsVesting 8
Roles 8
MoonLabsTokenLocker 9
Roles 9
MoonLabsLiquidityLocker 10
Roles 10
Diagnostics 11
UCF - Unimplemented Contract Function 13
Description 13
Recommendation 13
MWL - Misleading Whitelisted Logic 14
Description 14
Recommendation 14
DVI - Discount Value Inconsistency 15
Description 15
Recommendation 15
UCV - Uninitialized Contract Variables 16
Description 16
Recommendation 16
DSO - Data Structure Optimization 17
Description 17
Recommendation 17
CO - Code Optimization 18
Description 18
Recommendation 18
CR - Code Repetition 19
Description 19
Recommendation 19

MSC - Missing Sanity Checks 21
Description 21
Recommendation 21
MPV - Missing Percentages Validation 23
Description 23
Recommendation 23
PRA - Potential Re-entrance Attack 24
Description 24
Recommendation 24
CRO - Code Readability Optimizations 25
Description 25
Recommendation 25
TPP - Token Pair Prevalidation 26
Description 26
Recommendation 26
AAO - Accumulated Amount Overflow 27
Description 27
Recommendation 27
PTAI - Potential Transfer Amount Inconsistency 28
Description 28
Recommendation 28
L02 - State Variables could be Declared Constant 30
Description 30
Recommendation 30
L04 - Conformance to Solidity Naming Conventions 31
Description 31
Recommendation 32
L07 - Missing Events Arithmetic 33
Description 33
Recommendation 33
L08 - Tautology or Contradiction 34
Description 34
Recommendation 34
L11 - Unnecessary Boolean equality 35
Description 35
Recommendation 35
L13 - Divide before Multiply Operation 36
Description 36
Recommendation 36
L14 - Uninitialized Variables in Local Scope 37
Description 37
Recommendation 37

L16 - Validate Variable Setters 38
Description 38
Recommendation 38
L19 - Stable Compiler Version 39
Description 39
Recommendation 39
L20 - Succeeded Transfer Check 40
Description 40
Recommendation 40
Functions Analysis 41
Inheritance Graph 48
Flow Graph 49
Summary 50
Disclaimer 51
About Cyberscope 52

Review

Testing Deploy

Contract Name	Explorer
MoonLabsLiquidityLocker	https://testnet.bscscan.com/address/0x8A4A17Cb72f118c1eee262F5D1144678A48A3Fab
MoonLabsReferral	https://testnet.bscscan.com/address/0x78a9af1C8FB53d9999A8bB6e66A3E2436C25db7A
MoonLabsTokenLocker	https://testnet.bscscan.com/address/0x945149ba2F7569c1B1eAe068003f4b3747F3F07A
MoonLabsVesting	https://testnet.bscscan.com/address/0x47408f44B874c55A28A97Fc0802dD8fAdBC876B1
MoonLabsWhitelist	https://testnet.bscscan.com/address/0x2b97895c6Ac40429f04c57140771F7D743bEAd04

Audit Updates

Initial Audit	09 Mar 2023

Source Files

Filename	SHA256
IDEXRouter.sol	bff5d69c904ec69fff1c7a5e7c4c19088ddd c4f1342ea13f83317b0fe1136f3f
MoonLabsLiquidityLocker.sol	c021ea3c57bfbba72ee8097b0f50c50b75 041ef3ec8fcecbcedca640b783a490
MoonLabsReferral.sol	f3ef41718d2bd2a2fd45234063921d24a7 1bafafa0387eeab0b6b1c41484311b
MoonLabsTokenLocker.sol	feb646c4c147c0c44e0115a3055e12d9f1 491e9d7d86d47923c23fe1a0bece73
MoonLabsVesting.sol	b2cea2f09dfd04c62fea3649a435324f71a a9920ca9ff4dc72d6df531733fad8
MoonLabsWhitelist.sol	7cef1147938ea185aa39c1c9807e8e837 3896f8cf290901bf7f6ef8c1032bde1

Introduction

The Moonlab ecosystem consists of five contracts. Two utility contracts and three locker contracts.

MoonLabsWhitelist

The MoonLabsWhitelist contract is used for creating whitelists for Moon Labs products. Whitelisting a token allows users to waive all fees on related Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` is responsible for:

- Adding or removing a whitelisted address.
- Claim all contract deposited balance.
- Claim all `usdContract` token balance.

The users have the authority to:

- Purchase a whitelist with or without a discount code.
- Check the whitelisted address.

MoonLabsReferral

The MoonLabsReferral smart contract is used for creating and managing referral codes. It allows users to create referral codes for customers to use while purchasing Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Delete referral code.
- Add reserved codes.
- Assign or remove reserved codes.
- Add or remove addMoonLabsContract.
- Claim all contract deposited balance.

The user has the authority to:

- Check active codes.
- Get the address to the referral code.
- Get referral code to address.
- Add rewards earned for a code.
- Create, delete, transfer, and reserve referral codes.

MoonLabsVesting

The MoonLabsVesting Contract allows token owners to create ERC20 token locks. The Contract supports creating multiple vesting instances, choosing between linear and standard Locks, and transferring Locks to other addresses. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function. Lock creators cannot modify locks once they have been created, and withdraw owners cannot extend or change lock details.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.

MoonLabsTokenLocker

The MoonLabsTokenLocker contract is designed to allow users to create locks for ERC20 tokens. Lock creators can extend, transfer, add to, and split locks, but cannot unlock tokens prematurely. Users can create lock instances for the same token and choose either a linear or standard lock. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.

MoonLabsLiquidityLocker

The MoonLabsLiquidityLocker contract is responsible for creating liquidity locks for Uniswap-based AMM tokens. The main purpose of the contract is to allow users to create locks for selected wallets with the option to choose between standard or linear lock types. The lock type is determined by the start date, with the default being a standard lock. The locked tokens remain locked until their respective unlock date without any exceptions, and lock owners are not allowed to unlock them prematurely. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like importance, address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	UCF	Unimplemented Contract Function	Unresolved
Critical	MWL	Misleading Whitelisted Logic	Unresolved
Critical	DVI	Discount Value Inconsistency	Unresolved
Critical	UCV	Uninitialized Contract Variables	Unresolved
Minor / Informative	DSO	Data Structure Optimization	Unresolved
Minor / Informative	CO	Code Optimization	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	MSC	Missing Sanity Checks	Unresolved
Minor / Informative	MPV	Missing Percentages Validation	Unresolved
Minor / Informative	PRA	Potential Re-entrance Attack	Unresolved
Minor / Informative	CRO	Code Readability Optimizations	Unresolved
Minor / Informative	TPP	Token Pair Prevalidation	Unresolved

●	AAO	Accumulated Amount Overflow	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

UCF - Unimplemented Contract Function

Criticality	Critical
Location	MoonLabsLiquidityLocker.sol#L672
Status	Unresolved

Description

The contract has an unimplemented function that has been defined in the code. Specifically, the function `getClaimableTokens` has been declared but is missing the necessary code to execute its intended functionality. As a result, users will not be able to claim unlocked tokens.

```
/**
    * @notice Retrieve unlocked tokens for a lock instance
    * @param _nonce ID of desired lock instance
    * @return Number of unlocked tokens
    */
  function getClaimableTokens(uint64 _nonce) public view returns (uint)
{}
```

Recommendation

It is recommended to implement the missing function. Leaving unimplemented code in the application increases the risk of unintended behavior and could potentially create security vulnerabilities.

MWL - Misleading Whitelisted Logic

Criticality	Critical
Location	MoonLabsLiquidityLocker.sol#L376
Status	Unresolved

Description

The contract performs an incorrect if statement about the whitelisted user, which could lead to unintended behavior and potential security vulnerabilities.

The if statement checks if a user is whitelisted. But, as the comment states the contract expects not whitelisted addresses. As a result, whitelisted users will pay fees.

```
/// Check if the token is not whitelisted
if (whitelistContract.getIsWhitelisted(tokenAddress)) {
```

Recommendation

It is recommended to update the if statement to ensure that the correct conditions are checked. For instance,

```
if (!whitelistContract.getIsWhitelisted(tokenAddress)) {..}
```

DVI - Discount Value Inconsistency

Criticality	Critical
Location	MoonLabsWhitelist.sol#L
Status	Unresolved

Description

The contract applies a discount on transfer transactions based on the number of tokens being transferred. However, the require statement for the transfer and the transfer itself is inconsistent and result in an incorrect discount being applied to the transaction.

```
require(usdContract.balanceOf(msg.sender) >= costUSD - (costUSD *
codeDiscount) / 100, "Insignificant balance");
usdContract.transferFrom(msg.sender, address(this), (costUSD *
codeDiscount) / 100);
```

Recommendation

It is recommended to apply the same discount calculation logic in all cases. That means that the discount calculation logic should be consistent in every statement.

UCV - Uninitialized Contract Variables

Criticality	Critical
Location	MoonLabsWhitelist.sol#L58,59,90,93
Status	Unresolved

Description

The contract utilizes a variable in a transaction flow that is not initialized. The uninitialized variable can result in unexpected behavior, including the failure of the transaction or the loss of funds.

The variables `codeDiscount` and `codeCommission` are used to calculate the discount value for the transfer function, but they are not initialized. This can result in incorrect calculations or the transfer of incorrect amounts.

```
uint32 public codeDiscount;
require(usdContract.balanceOf(msg.sender) >= costUSD - (costUSD *
codeDiscount) / 100, "Insignificant balance");
usdContract.transferFrom(msg.sender, address(this), (costUSD *
codeDiscount) / 100);

uint32 public codeCommission;
distributeCommission(code, (costUSD * codeCommission) / 100);
```

Recommendation

It is recommended to initialize or calculate all the variables used in transaction flows. In this case, the variables `codeDiscount` and `codeCommission` should be initialized to a default value or set to the appropriate value before being used in any calculations.

DSO - Data Structure Optimization

Criticality	Minor / Informative
Location	MoonLabsReferral.sol#L48
Status	Unresolved

Description

The contract uses the valuable `reservedCodes` as an array. The business logic of the contract does not require to iterate this structure sequentially. Thus, unnecessary loops are produced that increase the required gas. Additionally, the stored values could be optimized to consume less memory.

```
string[] private reservedCodes;
```

Recommendation

The contract could use a data structure that provides instant access. For instance, a Set or a Map would fit better to the business logic of the contract. This way the time complexity will be reduced from o(n) to o(1).

The stored values could be stored as hashed values directly `keccak256(abi.encodePacked(code))`. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

CO - Code Optimization

Criticality	Minor / Informative
Location	MoonLabsReferral.sol
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.

There are code statements that could be optimized and redundant calculations.

The `keccak256(abi.encodePacked(""))` statement is utilized in multiple code segments.

The `depositAmount` and `totalDeposit` are the same. Hence the `MathUpgradeable.mulDiv` calculation is redundant.

```
keccak256(abi.encodePacked(""))

MathUpgradeable.mulDiv(amountSent, depositAmount, totalDeposit)
```

Recommendation

The team is advised to take into consideration these segments 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 statement `keccak256(abi.encodePacked(""))` could be defined as constant since it is utilized in multiple code segments.
- Redundant code statements could be removed.

CR - Code Repetition

Criticality	Minor / Informative
Location	MoonLabsTokenLocker.sol#133,173,216,259 MoonLabsVesting.sol#L119,158,200,242 MoonLabsLiquidityLocker.sol#L124,153,185,220,313,355,383,428
Status	Unresolved

Description

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.

```
uint totalDeposit;
for (uint32 i; i < lock.length; i++) {
    totalDeposit += lock[i].depositAmount;
}

uint64 _nonce = nonce;
/// Create a vesting instance for every struct in the lock array
for (uint64 i; i < lock.length; i++) {
    _nonce++;
    createVestingInstance(tokenAddress, lock[i], _nonce, amountSent,
totalDeposit);
}
nonce = _nonce;

require((depositAmount) <=
IERC20Upgradeable(tokenAddress).balanceOf(msg.sender), "Token
balance");

uint previousBal =
IERC20Upgradeable(tokenAddress).balanceOf(address(this));
/// Transfer tokens from sender to contract
transferTokensFrom(tokenAddress, msg.sender, depositAmount);
uint amountSent =
IERC20Upgradeable(tokenAddress).balanceOf(address(this)) - previousBal;
```

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 to reduce the complexity and size of the contract.

- The contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.
- The contract could utilize a safeTransfer library for common functionality that is widely used on the Moonlabs ecosystem.

MSC - Missing Sanity Checks

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L309,486 MoonLabsTokenLocker.sol#L353,475,527,766 MoonLabsLiquidityLocker.sol#L228
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 variables `newOwner`, `lock.ownerAddress`, `lock.withdrawAddress` and the `initializer` addresses are not properly sanitized.

```
function initialize(...){...}

function transferVestingOwnership(uint64 _nonce, address newOwner)
external {...}
function createVestingInstance(address tokenAddress, LockParams
calldata lock, uint64 _nonce, uint amountSent, uint totalDeposit)
private {...}

function setLockWithdrawAddress(uint64 _nonce, address _address)
external {...}
function splitLockETH(address to, address withdrawAddress, uint64
_nonce, uint amount) external payable {...}
function splitLockPercent(address to, address withdrawAddress, uint64
_nonce, uint amount) external {...}

function transferLockOwnership(uint64 _nonce, address newOwner)
external {...}
```

Recommendation

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

- The `initializer` addresses shouldn't be set to zero addresses.

- The `newOwner` address shouldn't be set to zero address.
- The `lock.withdrawAddress` address shouldn't be set to zero address.
- The `lock.ownerAddress` address shouldn't be set to zero address.

MPV - Missing Percentages Validation

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L388,396 MoonLabsTokenLocker.sol#L641,649 MoonLabsLiquidityLocker.sol#L560,568
Status	Unresolved

Description

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

The percentage `codeDiscount` and `codeCommission` are not properly sanitized.

```
function setCodeDiscount(uint8 _codeDiscount) external onlyOwner {
    codeDiscount = _codeDiscount;
}

function setCodeCommission(uint8 _codeCommission) external onlyOwner {
    codeCommission = _codeCommission;
}
```

Recommendation

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

- The `codeCommission` percentage should not be set to values greater than 100.
- The `codeDiscount` percentage should not be set to values greater than 100.

PRA - Potential Re-entrance Attack

Criticality	Minor / Informative
Location	MoonLabsTokenLocker.sol#L827 MoonLabsVesting.sol#L556 MoonLabsLiquidityLocker.sol#L717
Status	Unresolved

Description

The contract contains a function that can be vulnerable to reentrancy attacks. Specifically, the `distributeCommission` function allows a user to withdraw commissions from the contract balance, but the function does not prevent reentrancy attacks.

An attacker could potentially call the `createLockWithCodeEth` function multiple times that internally calls `distributeCommission` before the function completes, causing the contract to transfer `commission` repeatedly.

```
function distributeCommission(string memory code, uint commission)
private {
    /// Get referral code owner
    address payable to =
payable(referralContract.getAddressByCode(code));
    /// Send ether to code owner
    (bool sent, ) = to.call{ value: commission }("");
    require(sent, "Failed to send Ether");
    /// Log rewards in the referral contract
    referralContract.addRewardsEarned(code, commission);
}
```

Recommendation

To prevent reentrancy attacks, the contract should use a lock solution such as the Mutex or ReentrancyGuard pattern. These patterns use a lock variable to prevent multiple calls to the function while it is still being executed.

CRO - Code Readability Optimizations

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol MoonLabsReferral.sol MoonLabsVesting.sol MoonLabsTokenLocker.sol
Status	Unresolved

Description

The code in the contract is difficult to read and understand. The code repetition and the long code segments make it hard to follow the logic and understand the purpose of the code. This could increase the risk of introducing errors during maintenance or modification of the code.

```
require(referralContract.checkIfActive(code), "Invalid code");

require(msg.value == (ethLockPrice * lock.length - (((ethLockPrice *
codeDiscount) / 100) * lock.length)), "Incorrect price");
```

Recommendation

To improve code readability and reduce the risk of errors, the contract should include modifiers for common code segments and break long code segments into smaller segments or variables to make them more manageable. This will help other developers to better understand the code and maintain it more easily.

TPP - Token Pair Prevalidation

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol#L698 MoonLabsVesting.sol#L569 MoonLabsTokenLocker.sol#L807
Status	Unresolved

Description

The contract allows users to initiate swap transactions between two tokens without first checking if a token pair exists. This could result in the loss of funds if the contract is unable to find a liquidity pool for the specified token pair.

```
function handleBurns() private {
    /// Check if the threshold is met
    uint _burnMeter = burnMeter;
    if (burnMeter >= burnThreshold) {
      /// Buy tokenToBurn via Uniswap router and send to the dead
address
      address[] memory path = new address[](2);
      path[0] = routerContract.WETH();
      path[1] = address(tokenToBurn);

routerContract.swapExactETHForTokensSupportingFeeOnTransferTokens{
value: _burnMeter }(0, path,
0x000000000000000000000000000000000000dEaD, block.timestamp);
      _burnMeter = 0;
      burnMeter = _burnMeter;
    }
}
```

Recommendation

It is recommended to pre-validate that a token pair exists before allowing users to initiate swap transactions. A valid pair address should have token0, token1, factory

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol MoonLabsTokenLocker.sol MoonLabsVesting.sol
Status	Unresolved

Description

The contract is using the `noonce` variable to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint64 public nonce; /// Unique lock identifier
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L149 MoonLabsLiquidityLocker.sol#L691 MoonLabsTokenLocker.sol#L801 MoonLabsVesting.sol#L520
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
IERC20Upgradeable(tokenAddress).transfer(to, amount);

usdContract.transfer(referralContract.getAddressByCode(code),
commission);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L35,38,39
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.

```
uint public nonce
uint32 public codeDiscount
uint32 public codeCommission
```

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	MoonLabsWhitelist.sol#L51,65,82,91,116 MoonLabsVesting.sol#L32,33,345,374,413,421,429,437,445,453,461,469,477,486,519,537,568,627,693 MoonLabsTokenLocker.sol#L32,33,375,402,430,461,520,587,656,732,740,748,756,764,772,780,788,796,804,812,821,830,839,883,906,936,1032,1077 MoonLabsReferral.sol#L173,181,247,334 MoonLabsLiquidityLocker.sol#L31,32,295,321,353,355,410,412,471,535,605,613,621,629,637,645,653,661,669,677,685,694,703,712,745
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.

```
address _address
address _tokenToBurn
address _feeCollector
uint64 _nonce
address _routerAddress
address _referralAddress
uint _burnThreshold
uint _ethLockPrice
uint8 _codeDiscount
uint8 _codeCommission
uint8 _burnPercent
uint8 _percentLockPrice
uint _ethSplitPrice
uint _ethRelockPrice

...
```

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	MoonLabsVesting.sol#L438,479 MoonLabsTokenLocker.sol#L757,814 MoonLabsLiquidityLocker.sol#L630
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.

```
burnThreshold = _burnThreshold
burnPercent = _burnPercent
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L487 MoonLabsTokenLocker.sol#L822,831,840 MoonLabsLiquidityLocker.sol#L695,704,713
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(_percentLockPrice <= 10000, "Max percent")
require(_percentSplitPrice <= 10000, "Max percent")
require(_percentRelockPrice <= 10000, "Max percent")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	MoonLabsReferral.sol#L49,57,91,136,154,282
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(checkIfActive(code) == false, "Code in use")
require(checkIfReserved(_code) == false, "Code reserved")
require(checkIfActive(code) == true, "Code not in use")
require(checkIfReserved(_code) == false, "Code is reserved")
require(checkIfReserved(_code) == true, "Code not reserved")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L332 MoonLabsTokenLocker.sol#L362
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 prediction.

```
distributeCommission(
      code,
      (((ethLockPrice * codeCommission) / 100) * lock.length)
    )

distributeCommission(
      code,
      (((_ethLockPrice * codeCommission) / 100) * lock.length)
    )
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L121,139,171,197,234,251,298,315,382,630,642,699 MoonLabsTokenLocker.sol#L147,165,198,224,262,279,328,345,408,436,1037,1049,1061,1083
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.

```
uint32 i
uint64 i
uint64 timeElapsed
```

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	MoonLabsWhitelist.sol#L29 MoonLabsVesting.sol#L40,414 MoonLabsTokenLocker.sol#L40,733 MoonLabsLiquidityLocker.sol#L39,606
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.

```
usdAddress = _usdAddress
feeCollector = _feeCollector
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L8 MoonLabsVesting.sol#L10 MoonLabsTokenLocker.sol#L10 MoonLabsReferral.sol#L11 MoonLabsLiquidityLocker.sol#L9
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 ^0.8.17;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L55,71,108,129 MoonLabsVesting.sol#L606,620 MoonLabsTokenLocker.sol#L976,990 MoonLabsLiquidityLocker.sol#L810,824
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.

```
usdContract.transferFrom(msg.sender, address(this), costUSD)
usdContract.transferFrom(msg.sender, address(this), (costUSD *
codeDiscount) / 100)
usdContract.transferFrom(address(this), msg.sender,
usdContract.balanceOf(address(this)))
usdContract.transfer(referralContract.getAddressByCode(code),
commission)
IERC20Upgradeable(tokenAddress).transferFrom(from, address(this),
amount)
IERC20Upgradeable(tokenAddress).transfer(to, 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**
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsLiquidityLocker	Implementation	OwnableUpgradeable		
	initialize	Public	✓	initializer
	createLockWhitelist	External	✓	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner

	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	createLockInstance	Private	✓	
	getClaimableTokens	Public		-
	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	handleBurns	Private	✓	
	distributeCommission	Private	✓	
	deleteLockInstance	Private	✓	
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getCodeByAddress	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
	addRewardsEarnedUSD	External	✓	-

MoonLabsReferral	Implementation	IMoonLabsReferral, Ownable		
	createCode	External	✓	-
	deleteCode	External	✓	-
	deleteCodeOwner	External	✓	onlyOwner
	setCodeAddress	External	✓	-
	addReservedCodes	External	✓	onlyOwner
	assignReservedCode	External	✓	onlyOwner
	addMoonLabsContract	External	✓	onlyOwner
	removeMoonLabsContract	External	✓	onlyOwner
	addRewardsEarned	External	✓	-
	addRewardsEarnedUSD	External	✓	-
	getRewardsEarned	External		-
	getRewardsEarnedUSD	External		-
	getCodeByAddress	External		-
	getAddressByCode	External		-
	claimETH	External	✓	onlyOwner
	removeReservedCode	Public	✓	onlyOwner
	checkIfActive	Public		-
	checkIfReserved	Public		-
	upper	Private		
	_upper	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-

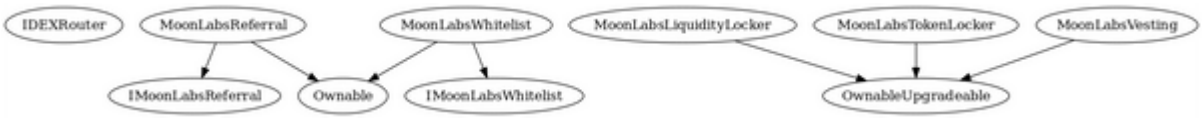
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsTokenLocker	Implementation	OwnableUpgradeable		
	initialize	Public	✓	initializer
	createLockWhitelist	External	✓	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	setLockWithdrawAddress	External	✓	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner

	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromWithdrawAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	getClaimableTokens	Public		-
	createLockInstance	Private	✓	
	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	handleBurns	Private	✓	
	distributeCommission	Private	✓	
	deleteLockInstance	Private	✓	
	calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsVesting	Implementation	OwnableUpgradeable		
	initialize	Public	✓	initializer
	createLockWhitelist	External	✓	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-

	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferVestingOwnership	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	getNonceFromWithdrawAddress	External		-
	getNonceFromTokenAddress	External		-
	getInstance	External		-
	getClaimableTokens	Public		-
	createVestingInstance	Private	✓	
	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	deleteVestingInstance	Private	✓	
	distributeCommission	Private	✓	
	handleBurns	Private	✓	
	calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-

	addRewardsEarnedUSD	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsWhitelist	Implementation	IMoonLabsWhitelist, Ownable		
		Public	✓	-
	purchaseWhitelist	External	✓	-
	purchaseWhitelistWithCode	External	✓	-
	ownerWhitelistAdd	External	✓	onlyOwner
	ownerWhitelistRemove	External	✓	onlyOwner
	claimETH	External	✓	onlyOwner
	claimUSD	External	✓	onlyOwner
	getIsWhitelisted	Public		-
	distributeCommission	Private	✓	

Inheritance Graph



Flow Graph



Summary

Moonlabs contract implements a utility, financial, and locker mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

MoonLabsStakingCards

March 2023

SHA256 fdac7fa4e7374b7b2b7ff4ba7903c23ef0c5b01e9482101ad16288388102a243

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 3
Introduction 5
Roles 5
Diagnostics 6
MT - Mints Tokens 7
Description 7
Recommendation 7
L02 - State Variables could be Declared Constant 8
Description 8
Recommendation 8
L04 - Conformance to Solidity Naming Conventions 9
Description 9
Recommendation 9
L14 - Uninitialized Variables in Local Scope 10
Description 10
Recommendation 10
Functions Analysis 11
Inheritance Graph 16
Flow Graph 17
Summary 18
Disclaimer 19
About Cyberscope 20

Review

Contract Name	MoonLabsStakingCards
Testing Deploy	https://testnet.bscscan.com/address/0x7e880151b52d2d95a5b3d854cb59ecd0cfdfbe1c
Symbol	TST

Audit Updates

Initial Audit	24 Mar 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC721/ERC721.sol	d0ae06bbac0c133b7891194ae955f624473d8572a7c6928747599f4043b01c04
@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol	e09a099c8828bb8eb545968ecca3f5f0fe418618eabd93b40befd6ec7e5a88be
@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol	ec7cb6a8cc0fdad9e3a707146b503d976595a880d24d62c99565787593f3660b
@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol	f16b861aa1f623ccc5e173f1a82d8cf45b678a7fb81e05478fd17eb2ccb7b37e
@openzeppelin/contracts/token/ERC721/IERC721.sol	c7703068bac02fe1cdf109e38faf10399c66eb411e4c9ae0d70c009eca4bf5ef
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	77f0f7340c2da6bb9edbc90ab6e7d3eb8e2ae18194791b827a3e8c0b11a09b43
@openzeppelin/contracts/utils/Address.sol	8160a4242e8a7d487d940814e5279d934e81f0436689132a4e73394bab084a6d
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5
@openzeppelin/contracts/utils/math/Math.sol	8059d642ec219d0b9b62fbc76912079529cf494cac988abe5e371f1168b29b0f
@openzeppelin/contracts/utils/Strings.sol	f81f11dca62dcd3e0895e680559676f4ba4f2e12a36bb0291d7ecbb6b983141f

contracts/MoonLabsStakingCards.sol	fdac7fa4e7374b7b2b7ff4ba7903c23ef0c5b01e9482101ad16288388102a243

Introduction

The MoonLabsStakingCards contract extends ERC721Enumerable and Ownable contracts from the OpenZeppelin library. It allows for the creation and management of non-fungible tokens (NFTs). The contract has a maximum supply of 500 NFTs, which can be minted by the contract owner only. The contract also provides a function for getting all the token IDs owned by a particular address, setting the base token URI, and claiming all ETH held in the contract. The tokenURI function returns the compiled token URI for a given token ID.

Roles

The contract consists of the owner role. The owner is responsible for the base URI configuration and for minting tokens.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	MT	Mints Tokens	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved

MT - Mints Tokens

Criticality	Minor / Informative
Location	MoonLabsStakingCards.sol#L50
Status	Unresolved

Description

The contract owner has the authority to mint tokens.

```
function ownerMint(address _to, uint amount) external onlyOwner {
  uint supply = totalSupply();
  require(supply + amount <= maxSupply, "Max supply reached");
  for (uint i = 1; i <= amount; i++) {
    _safeMint(_to, supply + i);
  }
}
```

Recommendation

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MoonLabsStakingCards.sol#L19,20
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 public baseExtension = ".json"
uint public maxSupply = 500
```

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	MoonLabsStakingCards.sol#L27,42,73
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.

```
address _owner
address _to
uint _id
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MoonLabsStakingCards.sol#L32
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.

```
uint i
```

Recommendation

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

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-

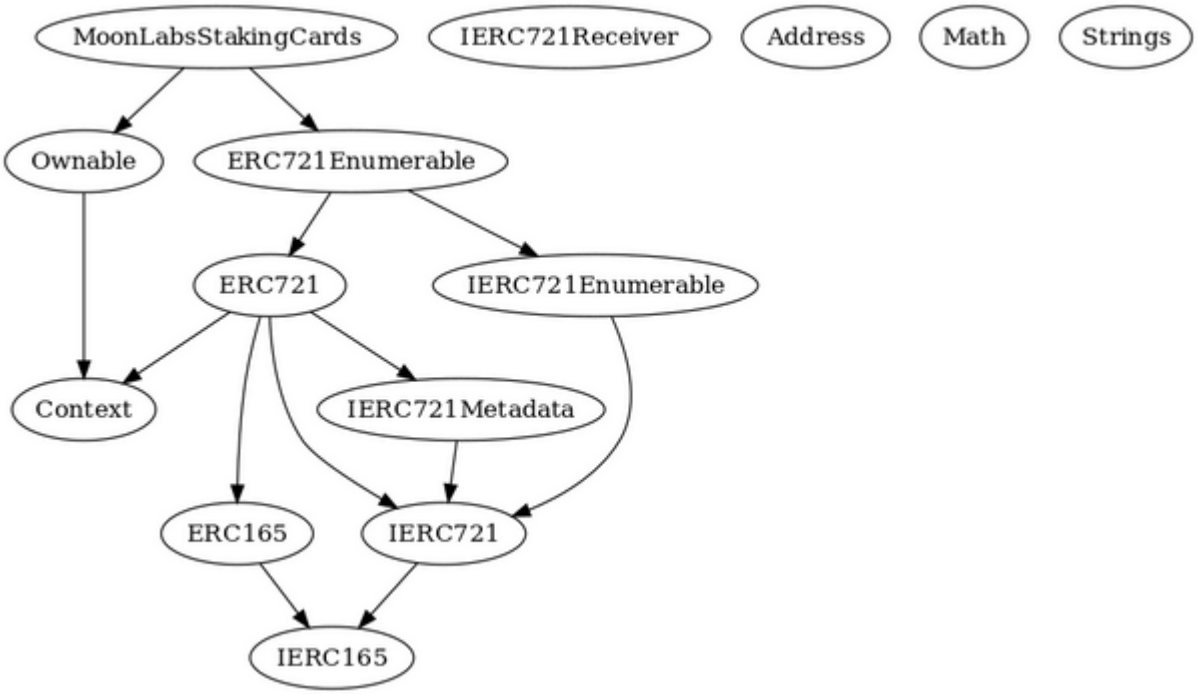
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_ownerOf	Internal		
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_requireMinted	Internal		
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	__unsafe_increaseBalance	Internal	✓	
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	
	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	

	_removeTokenFromAllTokensEnumeration	Private	✓	
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
IERC721Metadata	Interface	IERC721		
	name	External		-
	symbol	External		-
	tokenURI	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
Address	Library			
	isContract	Internal		

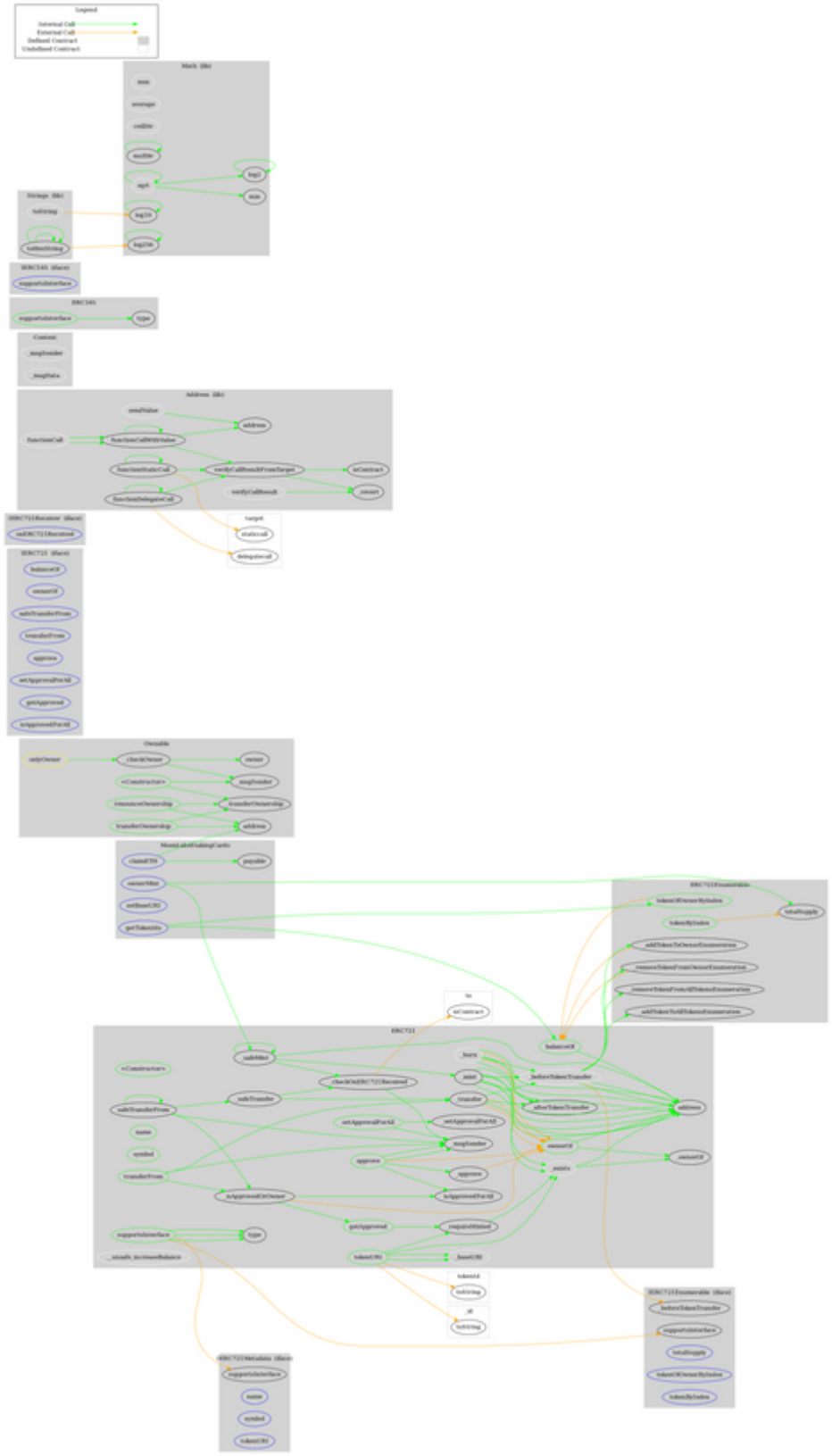
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
IERC165	Interface			
	supportsInterface	External		-
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		

	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
MoonLabsStakingCards	Implementation	ERC721Enumerable, Ownable		
		Public	✓	ERC721
	getTokenIds	External		-
	ownerMint	External	✓	onlyOwner
	setBaseURI	External	✓	onlyOwner
	claimETH	External	Payable	onlyOwner
	tokenURI	Public		-
	_baseURI	Internal		

Inheritance Graph



Flow Graph



Summary

MoonLabsStakingCards contract implements an nft mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

MoonLabs Token

March 2023

SHA256 aaa9a39a5f923d2ddb524d1957cd9963b400587d20c6a2156f11589bc087fdb0

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
- Audit Updates 3
- Source Files 4

Analysis 5
- ST - Stops Transactions 6
  - Description 6
  - Recommendation 6

Diagnostics 7
- RAV - Reentrancy Attack Vulnerability 8
  - Description 8
  - Recommendation 8
- RCS - Redundant Code Statement 9
  - Description 9
  - Recommendation 9
- MSC - Missing Sanity Check 10
  - Description 10
  - Recommendation 10
- ZD - Zero Division 11
  - Description 11
  - Recommendation 11
- PVC - Price Volatility Concern 12
  - Description 12
  - Recommendation 12
- DDP - Decimal Division Precision 13
  - Description 13
  - Recommendation 14
- MMSI - Max Mint Supply Inconsistency 15
  - Description 15
  - Recommendation 15
- MEE - Misleading Event Emission 16
  - Description 16
  - Recommendation 16
- L02 - State Variables could be Declared Constant 17
  - Description 17
  - Recommendation 17
- L04 - Conformance to Solidity Naming Conventions 18
  - Description 18
  - Recommendation 18

L07 - Missing Events Arithmetic 20
Description 20
Recommendation 20
L14 - Uninitialized Variables in Local Scope 21
Description 21
Recommendation 21
L16 - Validate Variable Setters 22
Description 22
Recommendation 22
Functions Analysis 23
Inheritance Graph 24
Flow Graph 25
Summary 26
Disclaimer 27
About Cyberscope 28

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
```

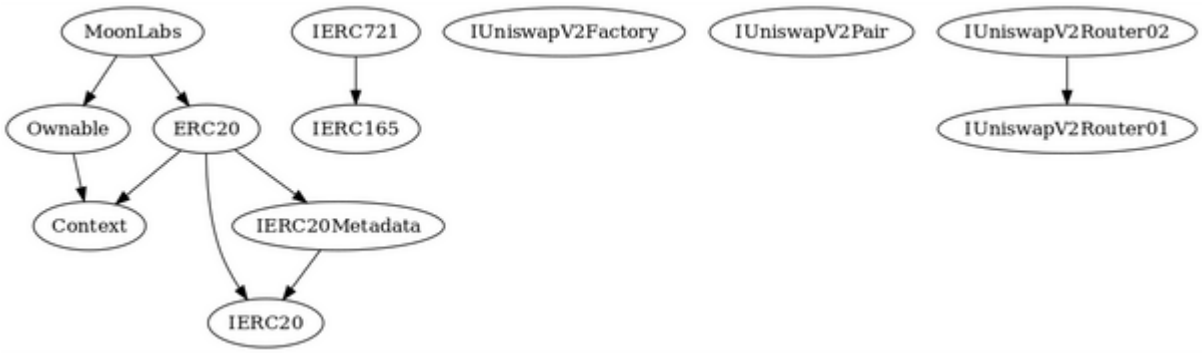
Recommendation

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.

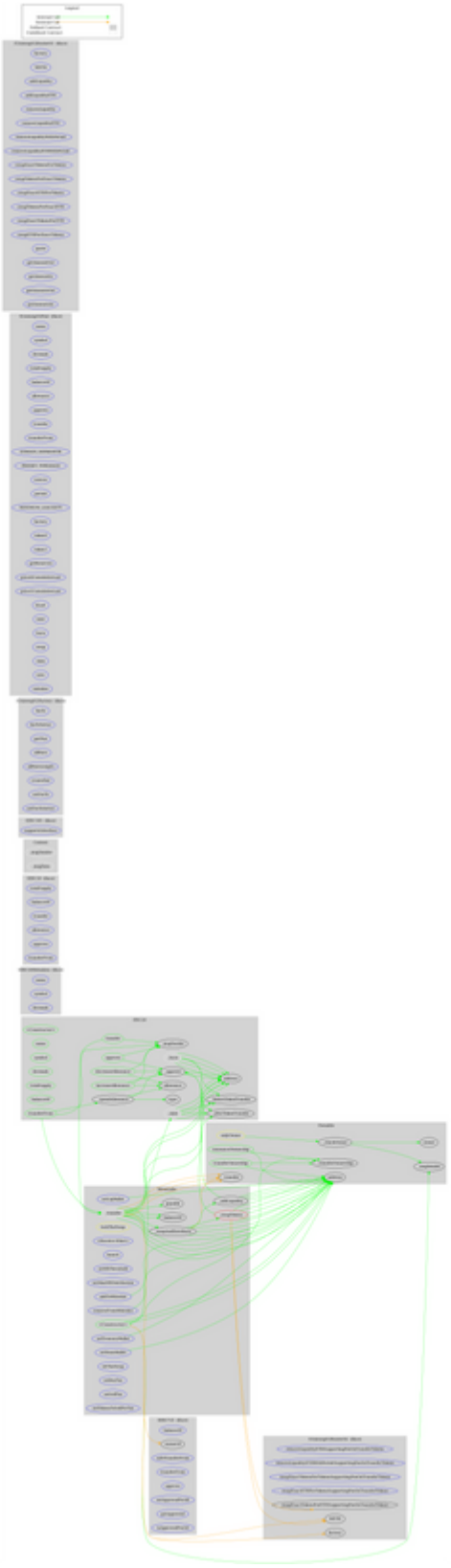
Functions Analysis

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

Cyberscope

Audit Report

Moonlabs

March 2023

SHA256 c3803461e1c537ca7cf72e504671bd17f2eeb2bcb42fab84b2101273133f17f6
1a7243a517c1125a3441c0a9663bc67c4373d0cea0a6f655f353a0057da2c229
46a4f16d522bf41f51494863831cefa4cee91c427b14775b84af196fbc11d0d8
11dc97a8973f8742d5e2df7cc8b9de21ee31f71f1c365afb6f47ab39a0f4c8f9
16b9b64c53c57591410f7c9261ca297da7d1f23f3700af85065c011e20b12866

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Testing Deploy 3
Introduction 4
MoonLabsWhitelist 4
Roles 4
MoonLabsReferral 5
Roles 5
MoonLabsVesting 6
Roles 6
MoonLabsTokenLocker 7
Roles 7
MoonLabsLiquidityLocker 8
Roles 8
Diagnostics 9
CO - Code Optimization 10
Description 10
Recommendation 10
MSC - Missing Sanity Checks 11
Description 11
Recommendation 11
PTAI - Potential Transfer Amount Inconsistency 12
Description 12
Recommendation 13
AAO - Accumulated Amount Overflow 14
Description 14
Recommendation 14
TPP - Token Pair Prevalidation 15
Description 15
Recommendation 15
L04 - Conformance to Solidity Naming Conventions 16
Description 16
Recommendation 17
L07 - Missing Events Arithmetic 18
Description 18
Recommendation 18
L16 - Validate Variable Setters 19
Description 19

Recommendation 19
Functions Analysis **20**
Inheritance Graph **28**
Flow Graph **29**
Summary **30**
Disclaimer **31**
About Cyberscope **32**

Review

Testing Deploy

Contract Name	Explorer
MoonLabsLiquidityLocker	https://testnet.snowtrace.io/address/0x83c9D3a5ADE7Ead2fE695E7d3626A99116556c3a
MoonLabsReferral	https://testnet.snowtrace.io/address/0xdCF65098B0873153A86C2408dFD012c4847fF299
MoonLabsTokenLocker	https://testnet.snowtrace.io/address/0xFE2a0790Bfbd566e002D976a1A89833BE6968cf7
MoonLabsVesting	https://testnet.snowtrace.io/address/0xfE84463C5eEB6623dd42A3911C63AeeEC64717B2
MoonLabsWhitelist	https://testnet.snowtrace.io/address/0xd0ac48Be795ADA54DC7260680A0379EDE6B54bad

Introduction

The Moonlab ecosystem consists of five contracts as upgradable proxies. Two utility contracts and three locker contracts.

MoonLabsWhitelist

The MoonLabsWhitelist contract is used for creating whitelists for Moon Labs products. Whitelisting a token allows users to waive all fees on related Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` is responsible for:

- Adding or removing a whitelisted address.
- Claim all contract deposited balance.
- Claim all `usdContract` token balance.

The users have the authority to:

- Purchase a whitelist with or without a discount code.
- Check the whitelisted address.

MoonLabsReferral

The MoonLabsReferral smart contract is used for creating and managing referral codes. It allows users to create referral codes for customers to use while purchasing Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Delete referral code.
- Add reserved codes.
- Assign or remove reserved codes.
- Add or remove addMoonLabsContract.
- Claim all contract deposited balance.

The user has the authority to:

- Check active codes.
- Get the address to the referral code.
- Get referral code to address.
- Add rewards earned for a code.
- Create, delete, transfer, and reserve referral codes.

MoonLabsVesting

The MoonLabsVesting Contract allows token owners to create ERC20 token locks. The Contract supports creating multiple vesting instances, choosing between linear and standard Locks, and transferring Locks to other addresses. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function. Lock creators cannot modify locks once they have been created, and withdraw owners cannot extend or change lock details.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.

MoonLabsTokenLocker

The MoonLabsTokenLocker contract is designed to allow users to create locks for ERC20 tokens. Lock creators can extend, transfer, add to, and split locks, but cannot unlock tokens prematurely. Users can create lock instances for the same token and choose either a linear or standard lock. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.

MoonLabsLiquidityLocker

The MoonLabsLiquidityLocker contract is responsible for creating liquidity locks for Uniswap-based AMM tokens. The main purpose of the contract is to allow users to create locks for selected wallets with the option to choose between standard or linear lock types. The lock type is determined by the start date, with the default being a standard lock. The locked tokens remain locked until their respective unlock date without any exceptions, and lock owners are not allowed to unlock them prematurely. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like importance, address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	MSC	Missing Sanity Checks	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	TPP	Token Pair Prevalidation	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L544
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 `depositAmount` and `totalDeposit` are the same. Hence the MathUpgradeable.mulDiv calculation is redundant.

```
MathUpgradeable.mulDiv(
    amountSent,
    depositAmount,
    totalDeposited
),
```

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.

- Redundant code statements could be removed.

MSC - Missing Sanity Checks

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol MoonLabsVesting.sol MoonLabsTokenLocker.sol
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.

```
function initialize(...){...}
function splitLockETH(address to, uint64 _nonce, uint amount ) external
payable {...}
function splitLockPercent(address to, uint64 _nonce, uint amount)
external {...}
function splitLockPercent(address to, address withdrawalAddress, uint64
_nonce, uint amount ) external {...}
```

Recommendation

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

- The `initializer` addresses shouldn't be set to zero addresses.
- The `to` addresses shouldn't be set to zero addresses.
- The `withdrawalAddress` addresses shouldn't be set to zero addresses.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol MoonLabsLiquidityLocker.sol#L854 MoonLabsVesting.sol#L651 MoonLabsTokenLocker.sol#L1022
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
usdContract.safeTransferFrom(
    address(this),
    msg.sender,
    usdContract.balanceOf(address(this))
);

IERC20Upgradeable(tokenAddress).safeTransferFrom(
    from,
    address(this),
    amount
);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

`Actual Transferred Amount = Balance After Transfer - Balance Before Transfer`

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol MoonLabsTokenLocker.sol MoonLabsVesting.sol
Status	Unresolved

Description

The contract is using the variable `nonce` to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint64 public nonce; /// Unique lock identifier
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

TPP - Token Pair Prevalidation

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol#L887 MoonLabsVesting.sol#L733 MoonLabsTokenLocker.sol#L1055
Status	Unresolved

Description

The contract allows users to initiate swap transactions between two tokens without first checking if a token pair exists. This could result in the loss of funds if the contract is unable to find a liquidity pool for the specified token pair.

```
function handleBurns() private {
    /// Check if the threshold is met
    uint _burnMeter = burnMeter;
    if (burnMeter >= burnThreshold) {
      /// Buy tokenToBurn via Uniswap router and send to the dead
address
      address[] memory path = new address[](2);
      path[0] = routerContract.WETH();
      path[1] = address(tokenToBurn);

routerContract.swapExactETHForTokensSupportingFeeOnTransferTokens{
value: _burnMeter }(0, path,
0x000000000000000000000000000000000000dEaD, block.timestamp);
      _burnMeter = 0;
      burnMeter = _burnMeter;
    }
}
```

Recommendation

It is recommended to pre-validate that a token pair exists before allowing users to initiate swap transactions. A valid pair address should have token0, token1, factory

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L37,62,80,108,117,126,134,143,173 MoonLabsVesting.sol#L44,45,262,298,299,344,353,362,371,379,387,396,405,413,422,455,473,658,735 MoonLabsTokenLocker.sol#L44,45,292,320,352,353,391,449,514,583,660,669,678,687,695,703,711,719,728,737,745,754,764,775,820,843,1074,1120 MoonLabsReferral.sol#L183,191,257,317 MoonLabsLiquidityLocker.sol#L35,36,256,281,314,316,365,367,420,481,552,561,570,579,587,595,603,611,620,629,637,646,656,667,701,805
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 _costUSD
address _usdAddress
address _address
uint8 _codeDiscount
uint8 _codeCommission
address _tokenToBurn
address _feeCollector
uint64 _nonce
address _routerAddress
address _referralAddress
uint _burnThreshold
uint _ethLockPrice
uint8 _burnPercent
uint16 _percentLockPrice

...
```

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	MoonLabsWhitelist.sol#L127,136,145 MoonLabsVesting.sol#L372,415 MoonLabsTokenLocker.sol#L688,747 MoonLabsLiquidityLocker.sol#L580,639
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.

```
costUSD = _costUSD
codeDiscount = _codeDiscount
codeCommission = _codeCommission
burnThreshold = _burnThreshold
burnPercent = _burnPercent
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L39 MoonLabsVesting.sol#L52 MoonLabsTokenLocker.sol#L52 MoonLabsLiquidityLocker.sol#L43
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.

```
usdAddress = _usdAddress
feeCollector = _feeCollector
```

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**
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactETHForTokens	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsLiquidityLocker	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		

	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	createLockInstance	Private	✓	

	transferAndCalculate	Private	✓	
	transferAndCalculateWithFee	Private	✓	
	getClaimableTokens	Public		-
	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	handleBurns	Private	✓	
	distributeCommission	Private	✓	nonReentrant
	deleteLockInstance	Private	✓	
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getCodeByAddress	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
	addRewardsEarnedUSD	External	✓	-
MoonLabsReferral	Implementation	Initializable, IMoonLabsReferral, OwnableUpgradeable		
	initialize	Public	✓	initializer
	createCode	External	✓	-
	deleteCode	External	✓	-
	deleteCodeOwner	External	✓	onlyOwner
	setCodeAddress	External	✓	-
	addReservedCodes	External	✓	onlyOwner
	assignReservedCode	External	✓	onlyOwner
	addMoonLabsContract	External	✓	onlyOwner
	removeMoonLabsContract	External	✓	onlyOwner
	addRewardsEarned	External	✓	-

	addRewardsEarnedUSD	External	✓	-
	getRewardsEarned	External		-
	getRewardsEarnedUSD	External		-
	getCodeByAddress	External		-
	getAddressByCode	External		-
	claimETH	External	✓	onlyOwner
	removeReservedCode	Public	✓	onlyOwner
	checkIfActive	Public		-
	upper	Private		
	_upper	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsTokenLocker	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-

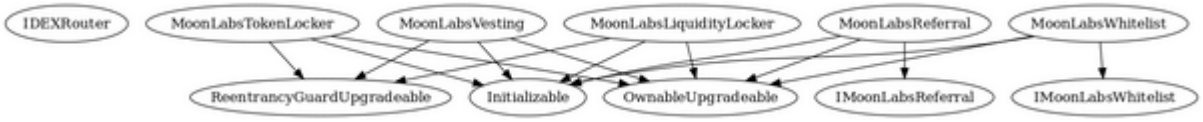
	setLockWithdrawalAddress	External	✓	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	getClaimableTokens	Public		-
	createLockInstances	Private	✓	
	calculateTotalDeposited	Private		
	transferAndCalculate	Private	✓	
	transferAndCalculateWithFee	Private	✓	

	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	handleBurns	Private	✓	
	distributeCommission	Private	✓	nonReentrant
	deleteLockInstance	Private	✓	
	calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsVesting	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferVestingOwnership	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner

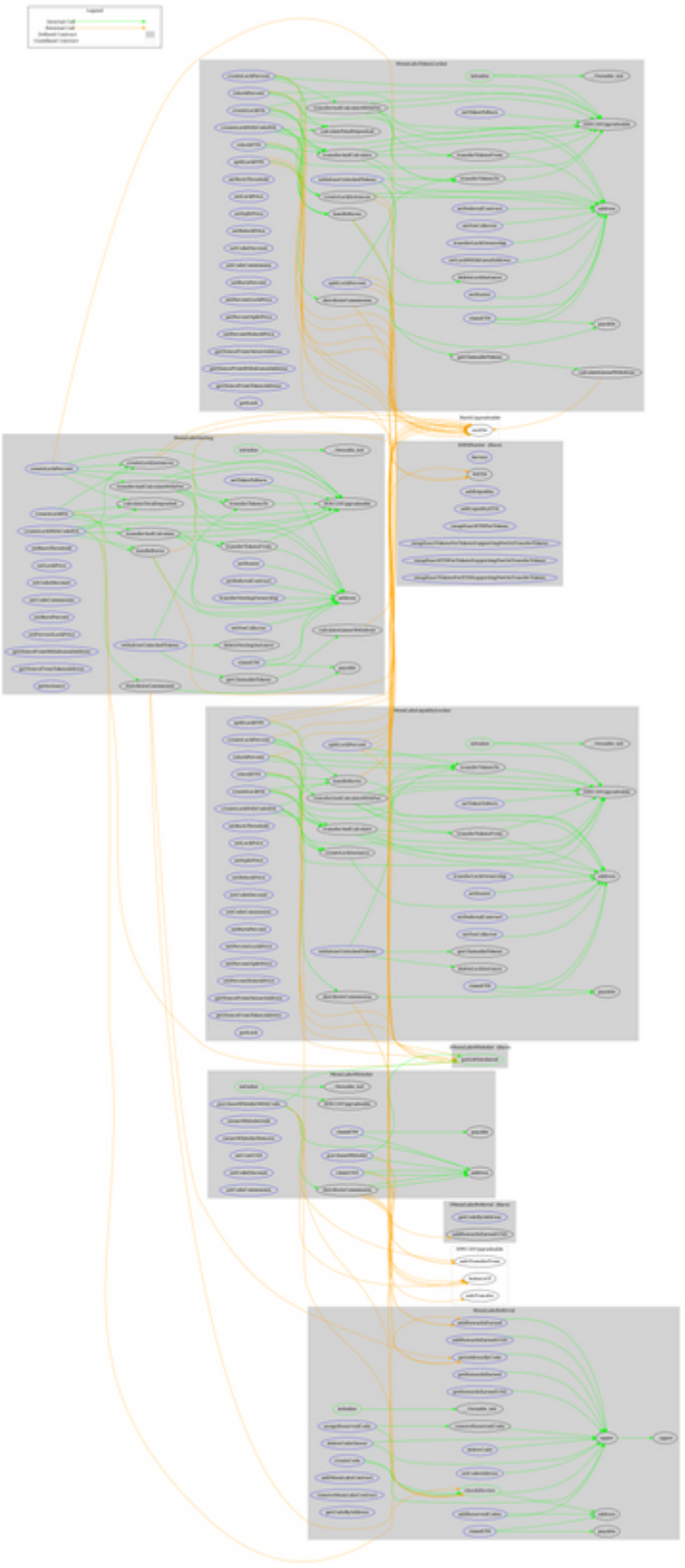
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getInstance	External		-
	getClaimableTokens	Public		-
	createLockInstances	Private	✓	
	calculateTotalDeposited	Private		
	transferAndCalculate	Private	✓	
	transferAndCalculateWithFee	Private	✓	
	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	deleteVestingInstance	Private	✓	
	distributeCommission	Private	✓	nonReentrant
	handleBurns	Private	✓	
	calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarnedUSD	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-

MoonLabsWhitelist	Implementation	Initializable, IMoonLabsWhitelist, OwnableUpgradeable		
	initialize	Public	✓	initializer
	purchaseWhitelist	External	✓	-
	purchaseWhitelistWithCode	External	✓	-
	ownerWhitelistAdd	External	✓	onlyOwner
	ownerWhitelistRemove	External	✓	onlyOwner
	setCostUSD	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	claimETH	External	✓	onlyOwner
	claimUSD	External	✓	onlyOwner
	getIsWhitelisted	Public		-
	distributeCommission	Private	✓	

Inheritance Graph



Flow Graph



Summary

Moonlabs contract implements a utility, financial, and locker mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

MoonLabsStakingCards

April 2023

SHA256 522ea07873e55484857dc679f7b42367026ba80985076e28e7008fac7703519c

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 3
Findings Breakdown 5
Introduction 6
Roles 6
Diagnostics 7
MT - Mints Tokens 8
Description 8
Recommendation 8
L02 - State Variables could be Declared Constant 9
Description 9
Recommendation 9
L04 - Conformance to Solidity Naming Conventions 10
Description 10
Recommendation 10
Functions Analysis 11
Inheritance Graph 18
Flow Graph 19
Summary 20
Disclaimer 21
About Cyberscope 22

Review

Contract Name	MoonLabsStakingCards
Testing Deploy	https://testnet.bscscan.com/address/0x6e8ce2a83e5ad4067c6ced0a7693c1d36cb5f088
Symbol	TsT

Audit Updates

Initial Audit	24 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabsd/v1/nft.pdf
Corrected Phase 2	06 Apr 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249a a4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC721/ERC721.sol	d0ae06bbac0c133b7891194ae955f62447 3d8572a7c6928747599f4043b01c04
@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol	e09a099c8828bb8eb545968ecca3f5f0fe4 18618eabd93b40befd6ec7e5a88be
@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol	ec7cb6a8cc0fdad9e3a707146b503d9765 95a880d24d62c99565787593f3660b
@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol	f16b861aa1f623ccc5e173f1a82d8cf45b67 8a7fb81e05478fd17eb2ccb7b37e
@openzeppelin/contracts/token/ERC721/IERC721.sol	c7703068bac02fe1cdf109e38faf10399c66 eb411e4c9ae0d70c009eca4bf5ef
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	77f0f7340c2da6bb9edbc90ab6e7d3eb8e 2ae18194791b827a3e8c0b11a09b43
@openzeppelin/contracts/utils/Address.sol	8160a4242e8a7d487d940814e5279d934e 81f0436689132a4e73394bab084a6d
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a2 3a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405 b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b30 64325801d73654847a5fb11c58b1e5
@openzeppelin/contracts/utils/math/Math.sol	8059d642ec219d0b9b62fbc76912079529 cf494cac988abe5e371f1168b29b0f

@openzeppelin/contracts/utils/Strings.sol	f81f11dca62dcd3e0895e680559676f4ba4f2e12a36bb0291d7ecbb6b983141f
contracts/MoonLabsStakingCards.sol	522ea07873e55484857dc679f7b42367026ba80985076e28e7008fac7703519c

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	3

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	3	0	0	0

Introduction

The MoonLabsStakingCards contract extends ERC721Enumerable and Ownable contracts from the OpenZeppelin library. It allows for the creation and management of non-fungible tokens (NFTs). The contract has a maximum supply of 500 NFTs, which can be minted by the contract owner only. The contract also provides a function for getting all the token IDs owned by a particular address, setting the base token URI, and claiming all ETH held in the contract. The tokenURI function returns the compiled token URI for a given token ID.

Roles

The contract consists of the owner role. The owner is responsible for the base URI configuration and for minting tokens.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	MT	Mints Tokens	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

MT - Mints Tokens

Criticality	Minor / Informative
Status	Unresolved

Description

The contract owner has the authority to mint tokens.

```
function ownerMint(address _to, uint amount) external onlyOwner
{
    uint supply = totalSupply();
    require(supply + amount <= MAX_SUPPLY, "Max supply
reached");
    for (uint i = 1; i <= amount; i++) {
      _safeMint(_to, supply + i);
    }
}
```

Recommendation

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/MoonLabsStakingCards.sol#L29
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.

```
nt nfts to. Only owner function.
```

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	contracts/MoonLabsStakingCards.sol#L29,39,54,87
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.

```
nt nfts to. Only owner function.

amount; i++) {

;
        r
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
		Public	✓	-
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-
	symbol	Public		-
	tokenURI	Public		-

	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_ownerOf	Internal		
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_requireMinted	Internal		
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	

	__unsafe_increaseBalance	Internal	✓	
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	
	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
IERC721Enumerable	Interface	IERC721		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
IERC721Metadata	Interface	IERC721		
	name	External		-
	symbol	External		-

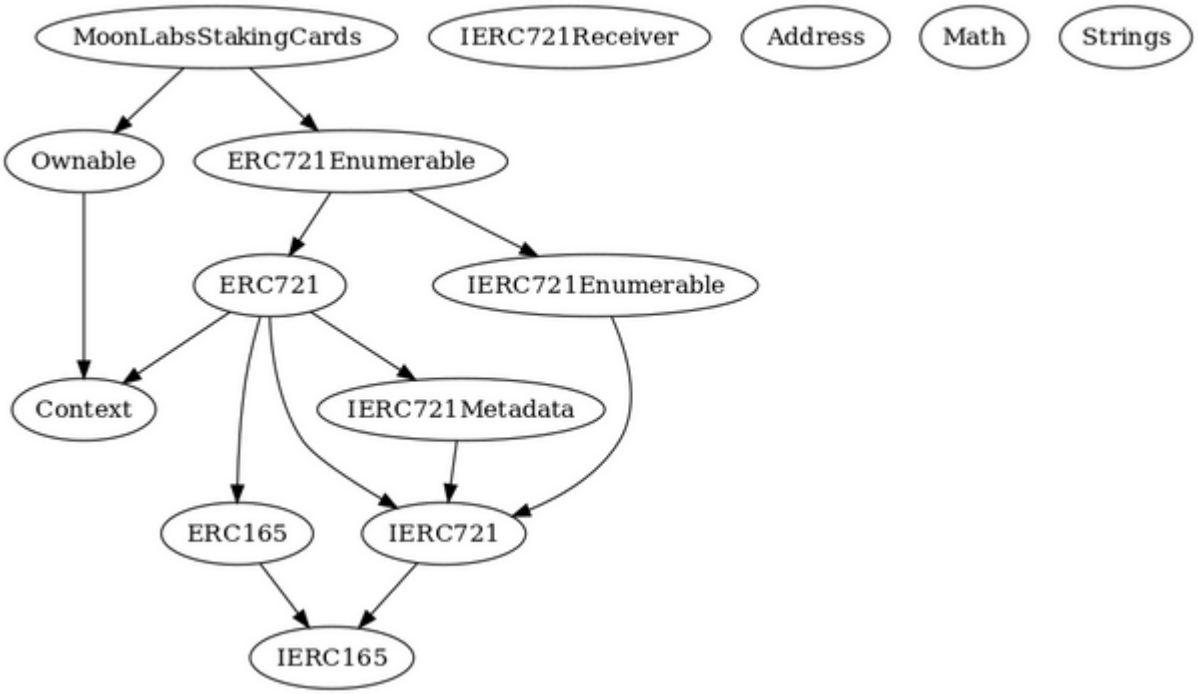
	tokenURI	External		-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	

	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
IERC165	Interface			
	supportsInterface	External		-
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		

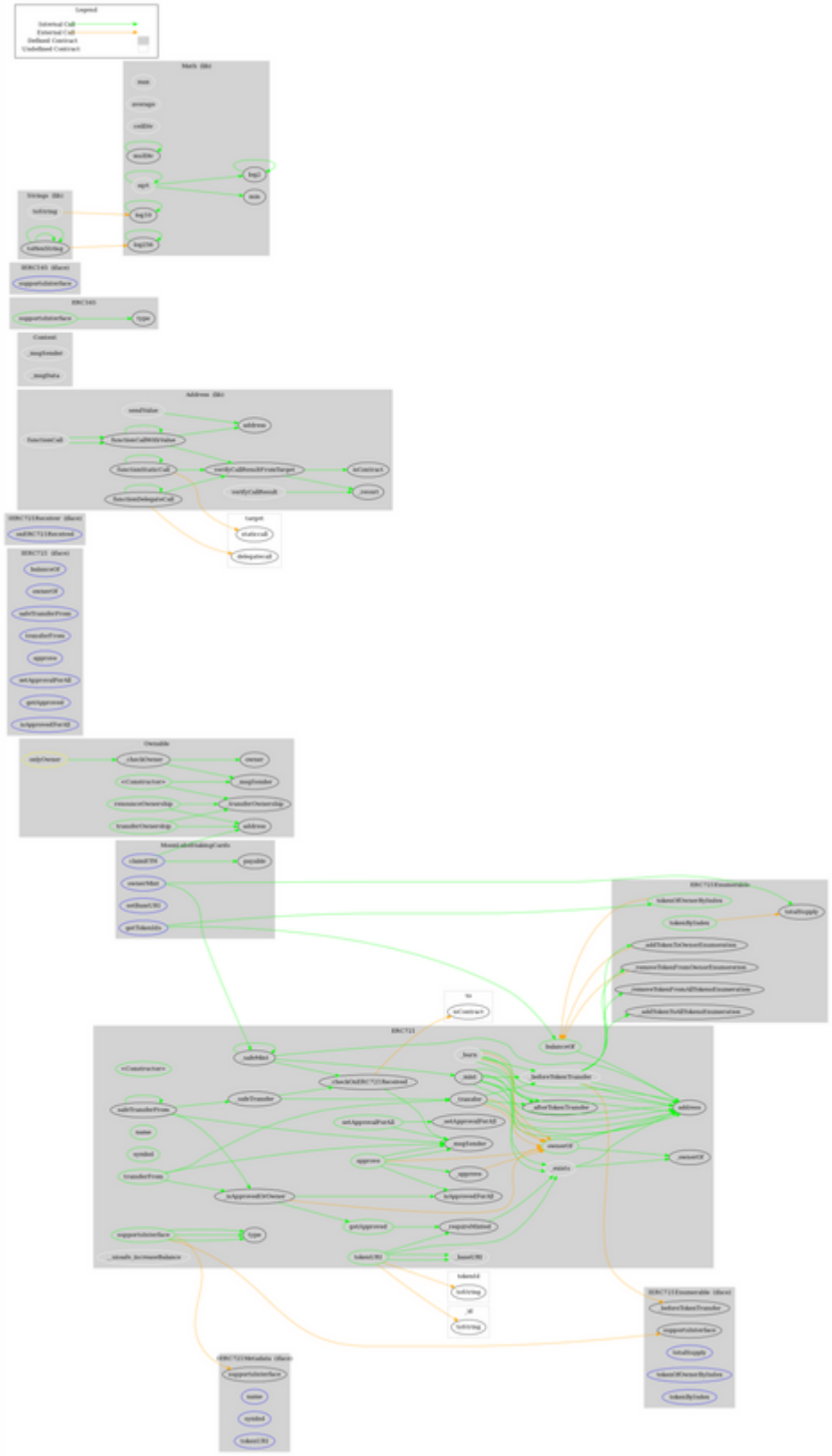
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
MoonLabsStakingCards	Implementation	ERC721Enumerable, Ownable		
		Public	✓	ERC721
	getTokenIds	External		-
	ownerMint	External	✓	onlyOwner
	setBaseURI	External	✓	onlyOwner

	claimETH	External	Payable	onlyOwner
	tokenURI	Public		-
	_baseURI	Internal		

Inheritance Graph



Flow Graph



Summary

MoonLabsStakingCards contract implements an nft mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

MoonLabs Token

April 2023

SHA256 503131e4c5848e3a6d6bed7c1c0cd113cf6b1327f41250916f1c7e2e52f42c4e

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
- Audit Updates 3
- Source Files 3

Analysis 4
Diagnostics 5
- RAV - Reentrancy Attack Vulnerability 6
  - Description 6
  - Recommendation 6
- RMC - Redundant Method Calls 7
  - Description 7
  - Recommendation 7
- MMSI - Max Mint Supply Inconsistency 9
  - Description 9
  - Recommendation 9
- DDP - Decimal Division Precision 10
  - Description 10
  - Recommendation 10
- MSC - Missing Sanity Check 12
  - Description 12
  - Recommendation 12
- RCS - Redundant Code Statement 13
  - Description 13
  - Recommendation 13
- L04 - Conformance to Solidity Naming Conventions 14
  - Description 14
  - Recommendation 14
- L07 - Missing Events Arithmetic 15
  - Description 15
  - Recommendation 15
- L14 - Uninitialized Variables in Local Scope 16
  - Description 16
  - Recommendation 16
- L16 - Validate Variable Setters 17
  - Description 17
  - Recommendation 17

Functions Analysis 18
Inheritance Graph 20
Flow Graph 21

Summary 22
Disclaimer 23
About Cyberscope 24

Review

Contract Name	MoonLabs
Testing Deploy	https://testnet.bscscan.com/address/0x3e1673aa20c037f9ffd9ca37a40ee5a1615ac5c1
Symbol	MLAB
Decimals	18
Total Supply	100.000.000

Audit Updates

Initial Audit	24 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabsd/v1/token.pdf
Corrected Phase 2	06 Apr 2023

Source Files

Filename	SHA256
MoonLabs.sol	503131e4c5848e3a6d6bed7c1c0cd113cf6b1327f41250916f1c7e2e52f42c4e

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
●	RAV	Reentrancy Attack Vulnerability	Unresolved
●	RMC	Redundant Method Calls	Unresolved
●	MMSI	Max Mint Supply Inconsistency	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	RCS	Redundant Code Statement	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	Minor / Informative
Location	MoonLabs.sol#L246
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 manipulate NFT ownership in order to receive more rewards.

A reentrance attack will occur if the user implements a receive callback, they will have the ability to execute any method again within the same execution thread.

For instance, a user could exploit the NFT reward mechanism by manipulating NFT ownership through buying and selling transactions near the same NFT reward index, leading to potential rewards manipulation.

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

Recommendation

The contract could disallow the use of contract addresses for transfer transactions and the contract could aggregate all rewards distributions for the NFTs and execute them all at once at the end of the function.

RMC - Redundant Method Calls

Criticality	Minor / Informative
Location	MoonLabs.sol#L317
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.

If there are no fees to distribute the methods `call` and `_addLiquidity` should not be called.

```
(treasuryWallet).call{
value: (ethBalance * (buyTax.treasuryTax + sellTax.treasuryTax)) /
    totalEthFee >
    0
    ? totalEthFee
    : 1
}("");
(teamWallet).call{
value: (ethBalance * (buyTax.teamTax + sellTax.teamTax)) /
    totalEthFee >
    0
    ? totalEthFee
    : 1
}("");

_addLiquidity(
    (addToLiquidityHalf),
    ((ethBalance * liquidityTax) / totalEthFee > 0 ? totalEthFee : 1) /
        2
);
```

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.

MMSI - Max Mint Supply Inconsistency

Criticality	Minor / Informative
Location	MoonLabs.sol#L261
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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	MoonLabs.sol#L317
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 taxes might not be splitted as expected.

```
uint addToLiquidityHalf = ((swapThreshold * liquidityTax) /
    totalTokenTax) / 2;
_addLiquidity((addToLiquidityHalf),
    ((ethBalance * liquidityTax) / totalEthFee > 0
    ? totalEthFee : 1) / 2 );

ethBalance * (buyTax.treasuryTax + sellTax.treasuryTax) /
    totalEthFee > 0 ? totalEthFee : 1
ethBalance * (buyTax.teamTax + sellTax.teamTax) / totalEthFee >
    0 ? totalEthFee : 1
ethBalance * buyTax.nftTax + sellTax.nftTax / totalEthFee >
    0 ? totalEthFee : 1
```

Recommendation

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

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	MoonLabs.sol#L242
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 be zero address.

RCS - Redundant Code Statement

Criticality	Minor / Informative
Location	MoonLabs.sol#L273
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);
```

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MoonLabs.sol#L121,125,131,137,142,147,152,157,197
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#L122,127,199
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#L228,266,309
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.

```
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#L71,72,73
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

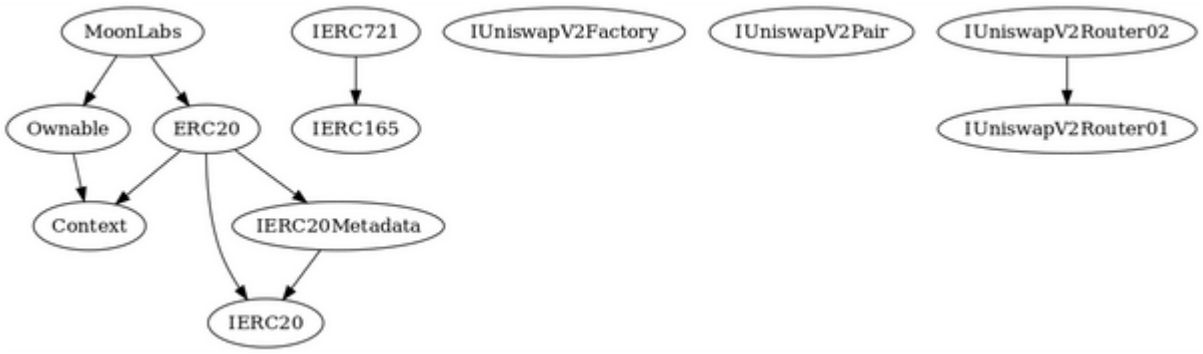
By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

Functions Analysis

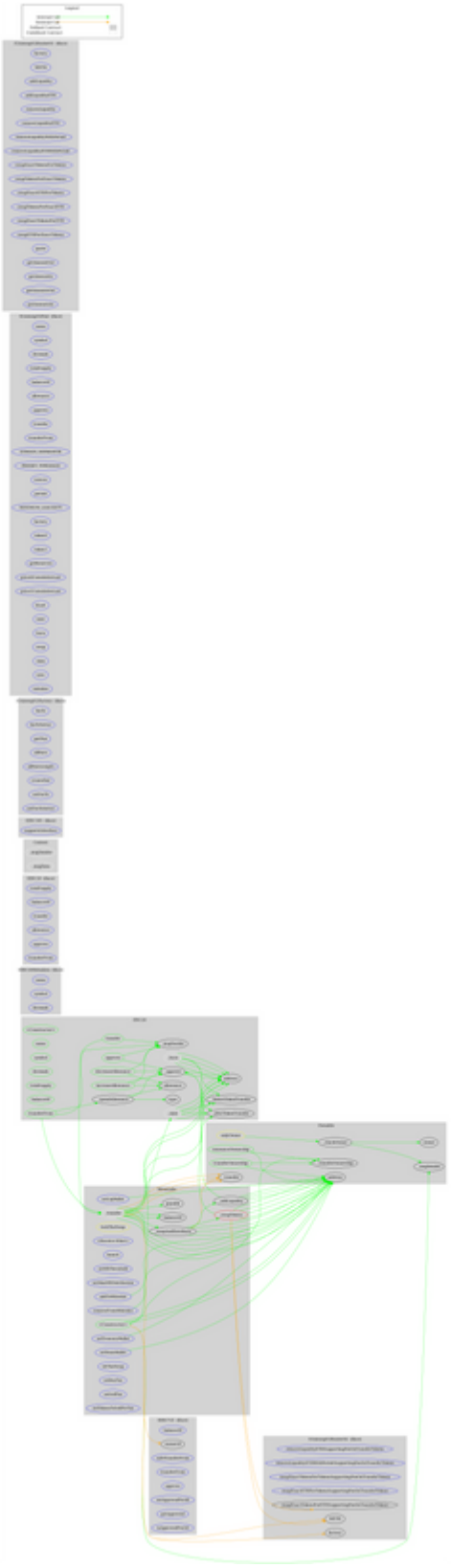
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MoonLabs	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20
		External	Payable	-
	launch	External	✓	onlyOwner
	setNftThreshold	External	✓	onlyOwner
	setMaxNftDistribution	External	✓	onlyOwner
	setTreasuryWallet	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	✓	nonReentrant
	_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. Moonlabs 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.

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

Cyberscope

Audit Report

Moonlabs

March 2023

SHA256 31cd30ebacbd1d3c336e67f58a0eba2e9b9fe9fa5716fdfe59e3317220dc4e9e
c3803461e1c537ca7cf72e504671bd17f2eeb2bcb42fab84b2101273133f17f6
6004b19aaf6be17db693c1488716e102ed024f315eebb991bca6b43f72d85851
a162ecd4fe206f07160297a15058b9e9b4b26e0b8da6a29cb44986c90b64964b
46a4f16d522bf41f51494863831cefa4cee91c427b14775b84af196fbc11d0d8

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Testing Deploy 4
Introduction 5
MoonLabsWhitelist 5
Roles 5
MoonLabsReferral 6
Roles 6
MoonLabsVesting 7
Roles 7
MoonLabsTokenLocker 8
Roles 8
MoonLabsLiquidityLocker 9
Roles 9
Diagnostics 10
TPP - Token Pair Prevalidation 11
Description 11
Recommendation 11
AAO - Accumulated Amount Overflow 12
Description 12
Recommendation 12
PTAI - Potential Transfer Amount Inconsistency 13
Description 13
Recommendation 14
MSC - Missing Sanity Checks 15
Description 15
Recommendation 15
CO - Code Optimization 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

L16 - Validate Variable Setters 20
Description 20
Recommendation 20
Functions Analysis **21**
Inheritance Graph **33**
Flow Graph **34**
Summary **35**
Disclaimer **36**
About Cyberscope **37**

Review

Audit Updates

Initial Audit	09 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v1/audit.pdf
Corrected Phase 2	20 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v2/audit.pdf
Corrected Phase 3	24 Mar 2023

Source Files

Filename	SHA256
IDEXRouter.sol	8d6779499e4bb704e8a5172278f4fcd82d299882cd4ef64ddd2fadcda24e202e
MoonLabsLiquidityLocker.sol	31cd30ebacbd1d3c336e67f58a0eba2e9b9fe9fa5716fdfe59e3317220dc4e9e
MoonLabsReferral.sol	c3803461e1c537ca7cf72e504671bd17f2eeb2bcb42fab84b2101273133f17f6
MoonLabsTokenLocker.sol	6004b19aaf6be17db693c1488716e102ed024f315eebb991bca6b43f72d85851
MoonLabsVesting.sol	a162ecd4fe206f07160297a15058b9e9b4b26e0b8da6a29cb44986c90b64964b
MoonLabsWhitelist.sol	46a4f16d522bf41f51494863831cefa4ce e91c427b14775b84af196fbc11d0d8

Testing Deploy

Contract Name	Explorer
MoonLabsLiquidityLocker	https://testnet.bscscan.com/address/0x467afa08EaE64d4cee8AFeD9aE0Fb5B5a4f53875#code
MoonLabsReferral	https://testnet.snowtrace.io/address/0xdCF65098B0873153A86C2408dFD012c4847fF299
MoonLabsTokenLocker	https://testnet.bscscan.com/address/0xf7b518558ca23454947D6880c0d904259f40C8c5
MoonLabsVesting	https://testnet.bscscan.com/address/0x2E9b0869603e3Ee86234491c72915C05Bb31e1e5
MoonLabsWhitelist	https://testnet.snowtrace.io/address/0xd0ac48Be795ADA54DC7260680A0379EDE6B54bad

Introduction

The Moonlab ecosystem consists of five contracts as upgradable proxies. Two utility contracts and three locker contracts.

MoonLabsWhitelist

The MoonLabsWhitelist contract is used for creating whitelists for Moon Labs products. Whitelisting a token allows users to waive all fees on related Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` is responsible for:

- Adding or removing a whitelisted address.
- Claim all contract deposited balance.
- Claim all `usdContract` token balance.

The users have the authority to:

- Purchase a whitelist with or without a discount code.
- Check the whitelisted address.

MoonLabsReferral

The MoonLabsReferral smart contract is used for creating and managing referral codes. It allows users to create referral codes for customers to use while purchasing Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Delete referral code.
- Add reserved codes.
- Assign or remove reserved codes.
- Add or remove addMoonLabsContract.
- Claim all contract deposited balance.

The user has the authority to:

- Check active codes.
- Get the address to the referral code.
- Get referral code to address.
- Add rewards earned for a code.
- Create, delete, transfer, and reserve referral codes.

MoonLabsVesting

The MoonLabsVesting Contract allows token owners to create ERC20 token locks. The Contract supports creating multiple vesting instances, choosing between linear and standard Locks, and transferring Locks to other addresses. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function. Lock creators cannot modify locks once they have been created, and withdraw owners cannot extend or change lock details.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.

MoonLabsTokenLocker

The MoonLabsTokenLocker contract is designed to allow users to create locks for ERC20 tokens. Lock creators can extend, transfer, add to, and split locks, but cannot unlock tokens prematurely. Users can create lock instances for the same token and choose either a linear or standard lock. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.

MoonLabsLiquidityLocker

The MoonLabsLiquidityLocker contract is responsible for creating liquidity locks for Uniswap-based AMM tokens. The main purpose of the contract is to allow users to create locks for selected wallets with the option to choose between standard or linear lock types. The lock type is determined by the start date, with the default being a standard lock. The locked tokens remain locked until their respective unlock date without any exceptions, and lock owners are not allowed to unlock them prematurely. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like importance, address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	TPP	Token Pair Prevalidation	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MSC	Missing Sanity Checks	Unresolved
●	CO	Code Optimization	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved

TPP - Token Pair Prevalidation

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol#L887MoonLabsVesting.sol#L733MoonLabsTokenLocker.sol#L1055
Status	Unresolved

Description

The contract allows users to initiate swap transactions between two tokens without first checking if a token pair exists. This could result in the loss of funds if the contract is unable to find a liquidity pool for the specified token pair.

```
function handleBurns() private {
    /// Check if the threshold is met
    uint _burnMeter = burnMeter;
    if (burnMeter >= burnThreshold) {
      /// Buy tokenToBurn via Uniswap router and send to the
dead address
      address[] memory path = new address[](2);
      path[0] = routerContract.WETH();
      path[1] = address(tokenToBurn);

routerContract.swapExactETHForTokensSupportingFeeOnTransferToke
ns{ value: _burnMeter }(0, path,
0x000000000000000000000000000000000000dEaD, block.timestamp);
      _burnMeter = 0;
      burnMeter = _burnMeter;
    }
}
```

Recommendation

It is recommended to pre-validate that a token pair exists before allowing users to initiate swap transactions. A valid pair address should have token0, token1, factory

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.solMoonLabsTokenLocker.solMoonLabsVesting.sol
Status	Unresolved

Description

The contract is using the variable `nonce` to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint64 public nonce; /// Unique lock identifier
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	MoonLabsWhitelist.solMoonLabsLiquidityLocker.sol#L854MoonLabsVesting.sol#L651MoonLabsTokenLocker.sol#L1022
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
usdContract.safeTransferFrom(
    address(this),
    msg.sender,
    usdContract.balanceOf(address(this))
);

IERC20Upgradeable(tokenAddress).safeTransferFrom(
    from,
    address(this),
    amount
);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

MSC - Missing Sanity Checks

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.solMoonLabsVesting.solMoonLabsTokenLocker.sol
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.

```
function initialize(...){...}
function splitLockETH(address to, uint64 _nonce, uint amount )
external payable {...}
function splitLockPercent(address to, uint64 _nonce, uint
amount) external {...}
function splitLockPercent(address to, address
withdrawalAddress, uint64 _nonce, uint amount ) external {...}
```

Recommendation

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

- The `initializer` addresses shouldn't be set to zero addresses.
- The `to` addresses shouldn't be set to zero addresses.
- The `withdrawalAddress` addresses shouldn't be set to zero addresses.

CO - Code Optimization

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L544
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 `depositAmount` and `totalDeposit` are the same. Hence the MathUpgradeable.mulDiv calculation is redundant.

```
MathUpgradeable.mulDiv(
    amountSent,
    depositAmount,
    totalDeposited
),
```

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.

- Redundant code statements could be removed.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L37,62,80,108,117,126,134,143,173MoonLabsVesting.sol#L42,43,255,286,330,339,348,357,365,373,382,391,399,408,441,459,627,698MoonLabsTokenLocker.sol#L42,43,285,315,316,317,355,392,447,509,578,655,664,673,682,690,698,706,714,723,732,740,749,758,768,813,836,1047,1092MoonLabsReferral.sol#L183,191,257,317MoonLabsLiquidityLocker.sol#L43,44,263,288,321,323,370,372,425,487,558,567,576,585,593,601,609,617,626,635,643,652,662,673,707,811
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 _costUSD
address _usdAddress
address _address
uint8 _codeDiscount
uint8 _codeCommission
address _tokenToBurn
address _feeCollector
uint64 _nonce
address _routerAddress
address _referralAddress
uint _burnThreshold
uint _ethLockPrice
uint8 _burnPercent
uint16 _percentLockPrice

...
```

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	MoonLabsWhitelist.sol#L127,136,145MoonLabsVesting.sol#L358,401MoonLabsTokenLocker.sol#L683,742MoonLabsLiquidityLocker.sol#L586,645
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.

```
costUSD = _costUSD
codeDiscount = _codeDiscount
codeCommission = _codeCommission
burnThreshold = _burnThreshold
burnPercent = _burnPercent
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L39MoonLabsVesting.sol#L50MoonLabsTokenLocker.sol#L50MoonLabsLiquidityLocker.sol#L51
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.

```
usdAddress = _usdAddress
feeCollector = _feeCollector
```

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**
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsLiquidityLocker	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuard		

		Upgradeable		
	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner

	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	createLockInstance	Private	✓	
	transferAndCalculate	Private	✓	
	transferAndCalculateWithFee	Private	✓	
	getClaimableTokens	Public		-
	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	handleBurns	Private	✓	

	distributeCommission	Private	✓	nonReentrant
	deleteLockInstance	Private	✓	
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getCodeByAddress	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
	addRewardsEarnedUSD	External	✓	-
MoonLabsReferral	Implementation	Initializable, IMoonLabsReferral, OwnableUpgradeable		
	initialize	Public	✓	initializer
	createCode	External	✓	-
	deleteCode	External	✓	-

	deleteCodeOwner	External	✓	onlyOwner
	setCodeAddress	External	✓	-
	addReservedCodes	External	✓	onlyOwner
	assignReservedCode	External	✓	onlyOwner
	addMoonLabsContract	External	✓	onlyOwner
	removeMoonLabsContract	External	✓	onlyOwner
	addRewardsEarned	External	✓	-
	addRewardsEarnedUSD	External	✓	-
	getRewardsEarned	External		-
	getRewardsEarnedUSD	External		-
	getCodeByAddress	External		-
	getAddressByCode	External		-
	claimETH	External	✓	onlyOwner
	removeReservedCode	Public	✓	onlyOwner
	checkIfActive	Public		-
	upper	Private		
	_upper	Private		

IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsTokenLocker	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-

	transferLockOwnership	External	✓	-
	setLockWithdrawalAddress	Public	✓	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner

	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	getClaimableTokens	Public		-
	createLockInstances	Private	✓	
	calculateTotalDeposited	Private		
	transferAndCalculate	Private	✓	
	transferAndCalculateWithFee	Private	✓	
	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	handleBurns	Private	✓	
	distributeCommission	Private	✓	nonReentrant
	deleteLockInstance	Private	✓	
	calculateLinearWithdraw	Private		

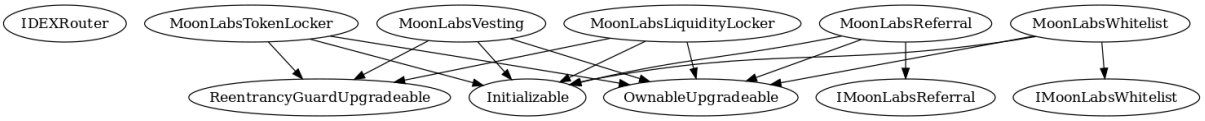
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsVesting	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-

	withdrawUnlockedTokens	External	✓	-
	transferVestingOwnership	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setTokenToBurn	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getInstance	External		-
	getClaimableTokens	Public		-
	createLockInstances	Private	✓	

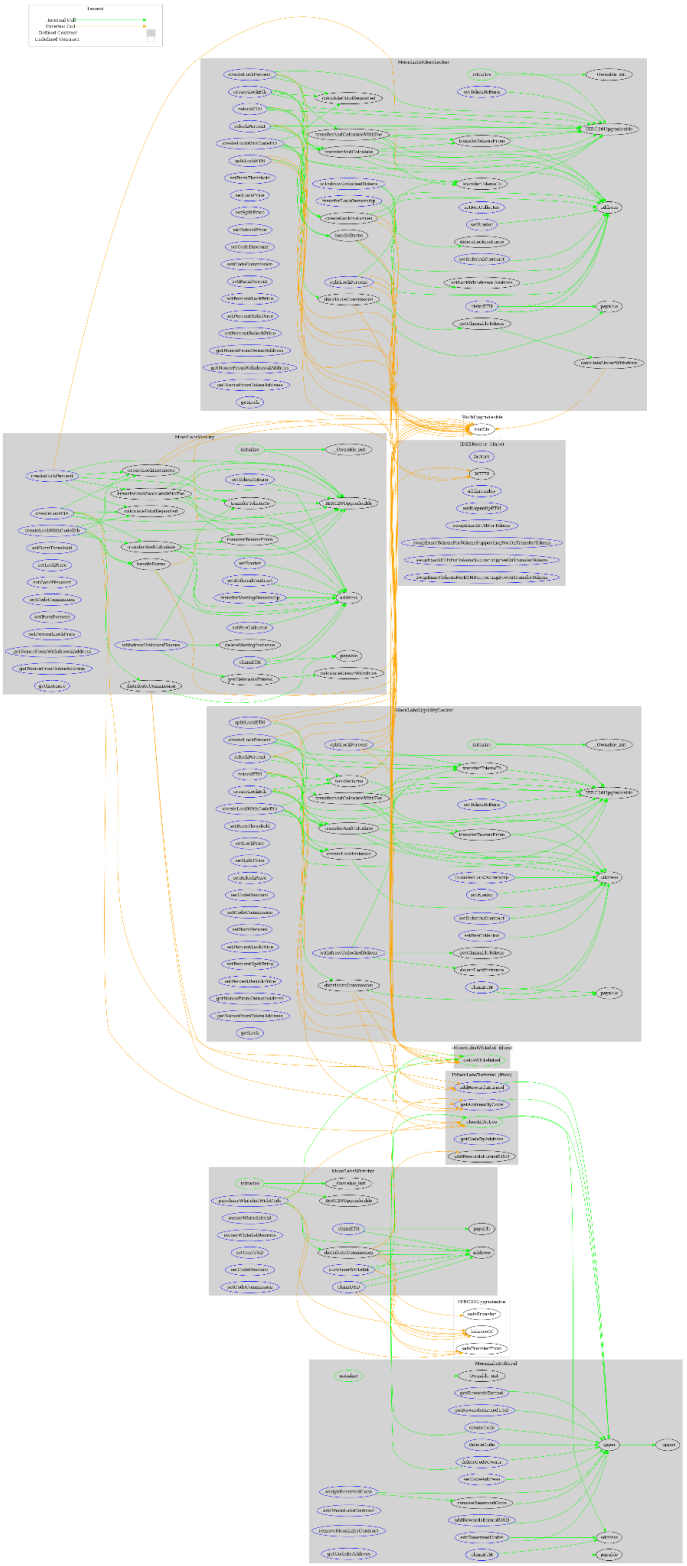
	calculateTotalDeposited	Private		
	transferAndCalculate	Private	✓	
	transferAndCalculateWithFee	Private	✓	
	transferTokensFrom	Private	✓	
	transferTokensTo	Private	✓	
	deleteVestingInstance	Private	✓	
	distributeCommission	Private	✓	nonReentrant
	handleBurns	Private	✓	
	calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarnedUSD	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-

MoonLabs Whitelist	Implementation	Initializable, IMoonLabsWhitelist, OwnableUpgradeable		
	initialize	Public	✓	initializer
	purchaseWhitelist	External	✓	-
	purchaseWhitelistWithCode	External	✓	-
	ownerWhitelistAdd	External	✓	onlyOwner
	ownerWhitelistRemove	External	✓	onlyOwner
	setCostUSD	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	claimETH	External	✓	onlyOwner
	claimUSD	External	✓	onlyOwner
	getIsWhitelisted	Public		-
	distributeCommission	Private	✓	

Inheritance Graph



Flow Graph



Summary

Moonlab's contracts implements a utility, financial, and locker mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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

Cyberscope

Audit Report

Moonlabs

March 2023

SHA256 e3580037e1fe6c148ee9090c746c093ba0a832ee2e17b9c7d0da377150f53307
b7b61dc5121be75e61487ac9d6e554c74c4ba7e268fe96998e484d1cec763b99
8f897febcfe9a904beb778789a70d9fb4d1ad994ba81c9fbcef0020139b61d74
dd4187851d23276f0580bc14bc026576141b7de603fb665c61f6ed629c92ea54
46a4f16d522bf41f51494863831cefa4cee91c427b14775b84af196fbc11d0d8

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 4
Testing Deploy 5
Findings Breakdown 6
Introduction 7
MoonLabsWhitelist 8
Roles 8
MoonLabsReferral 9
Roles 9
MoonLabsVesting 10
Roles 10
MoonLabsTokenLocker 11
Roles 11
MoonLabsLiquidityLocker 12
Roles 12
Diagnostics 13
PTM - Pair Token Mocking 14
Description 14
Recommendation 14
TPP - Token Pair Prevalidation 15
Description 15
Recommendation 15
AAO - Accumulated Amount Overflow 16
Description 16
Recommendation 16
PTAI - Potential Transfer Amount Inconsistency 17
Description 17
Recommendation 18
MSC - Missing Sanity Checks 19
Description 19
Recommendation 19
CO - Code Optimization 20
Description 20
Recommendation 20
L04 - Conformance to Solidity Naming Conventions 21
Description 21

Recommendation 22
L07 - Missing Events Arithmetic 23
Description 23
Recommendation 23
L12 - Using Variables before Declaration 25
Description 25
Recommendation 25
L14 - Uninitialized Variables in Local Scope 26
Description 26
Recommendation 26
L16 - Validate Variable Setters 27
Description 27
Recommendation 27
Functions Analysis 28
Inheritance Graph 46
Flow Graph 47
Summary 48
Disclaimer 49
About Cyberscope 50

Review

Audit Updates

Initial Audit	09 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v1/audit.pdf
Corrected Phase 2	20 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v2/audit.pdf
Corrected Phase 3	24 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/moonlabs/v3/audit.pdf
Corrected Phase 4	04 Apr 2023

Source Files

Filename	SHA256
IDEXRouter.sol	f6af1340f54f9d239f19282c0177481296b1b10699e094523136f381714afb30
MoonLabsLiquidityLocker.sol	e3580037e1fe6c148ee9090c746c093ba0a832ee2e17b9c7d0da377150f53307
MoonLabsLiquidityLockerAlt.sol	f4713e22d679da182da931033210140f6564bb64a9db52bf30fe8ffea507e619
MoonLabsTokenLocker.sol	b7b61dc5121be75e61487ac9d6e554c74c4ba7e268fe96998e484d1cec763b99
MoonLabsTokenLockerAlt.sol	6b6cba5974befb58457348f6710d2a6a39f4bd65970d59e4c19ab33ee8b1ca7c
MoonLabsVesting.sol	8f897febcfe9a904beb778789a70d9fb4d1ad994ba81c9fbcef0020139b61d74
MoonLabsVestingAlt.sol	fa6e7d5f1afe7b76f7280b70af90b1618f2fe186368f3724d411124a2f57f385
MoonLabsWhitelist.sol	dd4187851d23276f0580bc14bc026576141b7de603fb665c61f6ed629c92ea54
MoonLabsWhitelistAlt.sol	936a3727b62aaf96c10df43bb6db789d700e02e1a24a4dc1ad9fbaab7529f53e

Testing Deploy

Contract Name	Explorer
MoonLabsLiquidityLocker	https://testnet.bscscan.com/address/0x5a69C4666069d8142Ce4C90b0035c9956413b409
MoonLabsReferral	https://testnet.snowtrace.io/address/0xdCF65098B0873153A86C2408dFD012c4847fF299
MoonLabsTokenLocker	https://testnet.bscscan.com/address/0x6cF56Fa91322Da45cCFeAb2b333105e4613513D7
MoonLabsVesting	https://testnet.bscscan.com/address/0x4690a31FD346a596a3EBFaF5F43a872332a4CE42
MoonLabsWhitelist	https://testnet.bscscan.com/address/0x4690a31FD346a596a3EBFaF5F43a872332a4CE42

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

Introduction

The Moonlab ecosystem consists of five contracts as upgradable proxies. Two utility contracts and three locker contracts.

MoonLabsWhitelist

The MoonLabsWhitelist contract is used for creating whitelists for Moon Labs products. Whitelisting a token allows users to waive all fees on related Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` is responsible for:

- Adding or removing a whitelisted address.
- Claim all contract deposited balance.
- Claim all usdContract token balance.
- The users have the authority to:
- Purchase a whitelist with or without a discount code.
- Check the whitelisted address.
- Add or remove the blacklisted pairs.
- Configure contract address feeCollector, ReferralContract, Router, USDContract, MlabToken
- Set Mlab discount percent.

MoonLabsReferral

The MoonLabsReferral smart contract is used for creating and managing referral codes. It allows users to create referral codes for customers to use while purchasing Moon Labs products.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Delete referral code.
- Add reserved codes.
- Assign or remove reserved codes.
- Add or remove addMoonLabsContract.
- Claim all contract deposited balance.
- The user has the authority to:
- Check active codes.
- Get the address to the referral code.
- Get referral code to address.
- Add rewards earned for a code.
- Create, delete, transfer, and reserve referral codes.

MoonLabsVesting

The MoonLabsVesting Contract allows token owners to create ERC20 token locks. The Contract supports creating multiple vesting instances, choosing between linear and standard Locks, and transferring Locks to other addresses. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function. Lock creators cannot modify locks once they have been created, and withdraw owners cannot extend or change lock details.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Create a lock MLAB token.
- Get MLAB fee.

MoonLabsTokenLocker

The MoonLabsTokenLocker contract is designed to allow users to create locks for ERC20 tokens. Lock creators can extend, transfer, add to, and split locks, but cannot unlock tokens prematurely. Users can create lock instances for the same token and choose either a linear or standard lock. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like important, address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.
- Get claimable tokens.
- Create lock MLAB tokens.
- Relock MLAB tokens.
- split lock MLAB tokens.
- Get MLAB fee.

MoonLabsLiquidityLocker

The MoonLabsLiquidityLocker contract is responsible for creating liquidity locks for Uniswap-based AMM tokens. The main purpose of the contract is to allow users to create locks for selected wallets with the option to choose between standard or linear lock types. The lock type is determined by the start date, with the default being a standard lock. The locked tokens remain locked until their respective unlock date without any exceptions, and lock owners are not allowed to unlock them prematurely. The Contract also includes a feature to buy back and burn the native token, as well as a referral code system and whitelist function.

Roles

The contract roles consist of the owner role.

The `owner` role is responsible for:

- Claim all contract deposited balance.
- Configure contract parameters like importance, address, prices, and thresholds.

The user has the authority to:

- Create one or multiple vesting instances for a single token with fees or without fees or with a discount code.
- Transfer vesting instances ownership.
- Get nonces from address.
- Get the address from nonce.
- Get the lock tokens from the address.
- View claimable tokens.
- Withdraw unlocked tokens.
- Change withdrawal address.
- Relock or add tokens to an existing lock with or without fees.
- Divide a lock into multiple locks with or without fees.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PTM	Pair Token Mocking	Unresolved
●	TPP	Token Pair Prevalidation	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MSC	Missing Sanity Checks	Unresolved
●	CO	Code Optimization	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved

PTM - Pair Token Mocking

Criticality	Critical
Location	MoonLabsWhitelist.sol#L277
Status	Unresolved

Description

The contract does not properly validate token pairs, leaving it susceptible to potential token mocking attacks. Attackers could create fake tokens that mimic legitimate ones, potentially leading to unauthorized transactions and loss of funds for the contract and its users.

```
function getIsWhitelisted(address _address) public view override
returns (bool) {
    if (tokenToWhitelist[_address]) return true;
    /// Check for v2 pairs
    if (_checkV2Pair(_address)) return true;
    /// Check for v3 pools
    if (_checkV3Pool(_address)) return true;
    return false;
}
```

Recommendation

It is recommended to implement robust validation checks to ensure that only legitimate token pairs are used in the contract. For Instance, to ensure that the pair address is legitimate, the contract could leverage a trusted address to access the factory contract's `getPair` function and validate that the returned pair address is identical to the provided argument address.

TPP - Token Pair Prevalidation

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol#L887 MoonLabsVesting.sol#L733 MoonLabsTokenLocker.sol#L1055
Status	Unresolved

Description

The contract allows users to initiate swap transactions between two tokens without first checking if a token pair exists. This could result in the loss of funds if the contract is unable to find a liquidity pool for the specified token pair.

```
function handleBurns() private {
    /// Check if the threshold is met
    uint _burnMeter = burnMeter;
    if (burnMeter >= burnThreshold) {
      /// Buy tokenToBurn via Uniswap router and send to the
dead address
      address[] memory path = new address[](2);
      path[0] = routerContract.WETH();
      path[1] = address(tokenToBurn);

routerContract.swapExactETHForTokensSupportingFeeOnTransferToke
ns{ value: _burnMeter }(0, path,
0x000000000000000000000000000000000000dEaD, block.timestamp);
      _burnMeter = 0;
      burnMeter = _burnMeter;
    }
}
```

Recommendation

It is recommended to pre-validate that a token pair exists before allowing users to initiate swap transactions. A valid pair address should have token0, token1, factory

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol MoonLabsTokenLocker.sol MoonLabsVesting.sol
Status	Unresolved

Description

The contract is using the variable `nonce` to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
uint64 public nonce; /// Unique lock identifier
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol MoonLabsLiquidityLocker.sol#L854 MoonLabsVesting.sol#L651 MoonLabsTokenLocker.sol#L1022
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
usdContract.safeTransferFrom(
    address(this),
    msg.sender,
    usdContract.balanceOf(address(this))
);

IERC20Upgradeable(tokenAddress).safeTransferFrom(
    from,
    address(this),
    amount
);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

`Actual Transferred Amount = Balance After Transfer - Balance Before Transfer`

MSC - Missing Sanity Checks

Criticality	Minor / Informative
Location	MoonLabsLiquidityLocker.sol MoonLabsVesting.sol MoonLabsTokenLocker.sol
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.

```
function initialize(...){...}
function splitLockETH(address to, uint64 _nonce, uint amount )
external payable {...}
function splitLockPercent(address to, uint64 _nonce, uint
amount) external {...}
function splitLockPercent(address to, address
withdrawalAddress, uint64 _nonce, uint amount ) external {...}
```

Recommendation

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

- The `initializer` addresses shouldn't be set to zero addresses.
- The `to` addresses shouldn't be set to zero addresses.
- The `withdrawalAddress` addresses shouldn't be set to zero addresses.

CO - Code Optimization

Criticality	Minor / Informative
Location	MoonLabsVesting.sol#L544
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 `depositAmount` and `totalDeposit` are the same. Hence the MathUpgradeable.mulDiv calculation is redundant.

```
MathUpgradeable.mulDiv(
    amountSent,
    depositAmount,
    totalDeposited
),
```

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.

- Redundant code statements could be removed.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L52,85,98,114,133,143,153,163,173,181,190,199,208,217,226,235,244,264,268,277MoonLabsVesting.sol#L62,63,337,373,374,419,428,437,446,454,462,471,481,491,500,533,572MoonLabsTokenLocker.sol#L62,63,366,401,402,403,441,478,502,532,568,598,638,686,695,704,713,721,729,737,745,754,764,774,783,793,804,849,893MoonLabsLiquidityLocker.sol#L60,61,313,343,373,399,421,449,482,511,547,592,601,610,619,627,635,643,651,660,670,680,689,699,710,744,782
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.

```
ommission; //
nt in the percenta
 applied to the custo
d token to globa

hitelisted
    r

st
    tokenToWh
s) external only
dress _address)
int _costUSD) ex

...
```

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	MoonLabsWhitelist.sol#L174,183,201,246MoonLabsVesting.sol#L447,484,493MoonLabsTokenLocker.sol#L714,767,776MoonLabsLiquidityLocker.sol#L620,673,682
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.

```
to the code owner
contract address. Owner only
ice Set the USD token address. O

ew returns (bool) {
    return pairToBlack

...

ss
    ) external view ret
 setRelockPrice(uint _ethReloc

onlyOwner {
        requir
```

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.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L349,354,375,380
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
= IUniswapV3Po

) {
      retur
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MoonLabsWhitelist.sol#L349,354,375,380
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.

```
= IUniswapV3Po

) {
      retur
```

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	MoonLabsWhitelist.sol#L54MoonLabsVesting.sol#L70MoonLabsTokenLocker.sol#L70MoonLabsLiquidityLocker.sol#L68
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.

```
al code owner, represented i
EI for buying and burning ml
 ethSplitPrice; /// Price in
c ethSplitPrice; /// Price i
```

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**
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactETHForTokens	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	getAmountsOut	External		-
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			

	getIsWhitelisted	External		-
MoonLabsLiquidityLocker	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockMLAB	External	Payable	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	nukeLock	External	✓	-
	relockMLAB	External	Payable	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockMLAB	External	Payable	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner

	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setMlabToken	External	✓	onlyOwner
	setMlabDiscountPercent	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	getMLABFee	Public		-
	getClaimableTokens	Public		-
	_buyWithMLAB	Private	✓	
	_relock	Private	✓	
	_splitLock	Private	✓	
	_createLock	Private	✓	
	_transferAndCalculate	Private	✓	
	_transferAndCalculateWithFee	Private	✓	
	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	

	_handleBurns	Private	✓	
	_distributeCommission	Private	✓	nonReentrant
	_deleteLockInstance	Private	✓	
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsLiquidityLockerAlt	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-

	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setBurnCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	getClaimableTokens	Public		-
	_createLock	Private	✓	
	_transferAndCalculate	Private	✓	
	_transferAndCalculateWithFee	Private	✓	

	_relock	Private	✓	
	_splitLock	Private	✓	
	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	
	_handleBurns	Private	✓	
	_distributeCommission	Private	✓	nonReentrant
	_deleteLockInstance	Private	✓	
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsTokenLocker	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockMLAB	External	✓	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-

	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	setLockWithdrawalAddress	Public	✓	-
	relockMLAB	External	Payable	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockMLAB	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setMlabToken	External	✓	onlyOwner
	setMlabDiscountPercent	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner

	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	getMLABFee	Public		-
	getClaimableTokens	Public		-
	_buyWithMLAB	Private	✓	
	_relock	Private	✓	
	_splitLock	Private	✓	
	_createLocks	Private	✓	
	_calculateTotalDeposited	Private		
	_transferAndCalculate	Private	✓	
	_transferAndCalculateWithFee	Private	✓	
	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	
	_handleBurns	Private	✓	
	_distributeCommission	Private	✓	nonReentrant
	_deleteLockInstance	Private	✓	
	_calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			

	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsTokenLockerAlt	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-
	transferLockOwnership	External	✓	-
	setLockWithdrawalAddress	Public	✓	-
	relockETH	External	Payable	-
	relockPercent	External	✓	-
	splitLockETH	External	Payable	-
	splitLockPercent	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner

	setBurnCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setSplitPrice	External	✓	onlyOwner
	setRelockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	setPercentSplitPrice	External	✓	onlyOwner
	setPercentRelockPrice	External	✓	onlyOwner
	getNonceFromOwnerAddress	External		-
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getLock	External		-
	getClaimableTokens	Public		-
	_relock	Private	✓	
	_splitLock	Private	✓	
	_createLocks	Private	✓	
	_calculateTotalDeposited	Private		
	_transferAndCalculate	Private	✓	
	_transferAndCalculateWithFee	Private	✓	

	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	
	_handleBurns	Private	✓	
	_distributeCommission	Private	✓	nonReentrant
	_deleteLockInstance	Private	✓	
	_calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsVesting	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockMLAB	External	✓	-
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-

	withdrawUnlockedTokens	External	✓	-
	transferVestingOwnership	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setMlabToken	External	✓	onlyOwner
	setMlabDiscountPercent	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getInstance	External		-
	getMLABFee	Public		-
	getClaimableTokens	Public		-
	_buyWithMLAB	Private	✓	
	_createLocks	Private	✓	
	_calculateTotalDeposited	Private		
	_transferAndCalculate	Private	✓	
	_transferAndCalculateWithFee	Private	✓	

	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	
	_deleteVestingInstance	Private	✓	
	_distributeCommission	Private	✓	nonReentrant
	_handleBurns	Private	✓	
	_calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsVestingAlt	Implementation	Initializable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	createLockPercent	External	✓	-
	createLockEth	External	Payable	-
	createLockWithCodeEth	External	Payable	-
	withdrawUnlockedTokens	External	✓	-

	transferVestingOwnership	External	✓	-
	claimETH	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setBurnCollector	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setBurnThreshold	External	✓	onlyOwner
	setLockPrice	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setBurnPercent	External	✓	onlyOwner
	setPercentLockPrice	External	✓	onlyOwner
	getNonceFromWithdrawalAddress	External		-
	getNonceFromTokenAddress	External		-
	getInstance	External		-
	getClaimableTokens	Public		-
	_createLocks	Private	✓	
	_calculateTotalDeposited	Private		
	_transferAndCalculate	Private	✓	
	_transferAndCalculateWithFee	Private	✓	
	_transferTokensFrom	Private	✓	
	_transferTokensTo	Private	✓	
	_deleteVestingInstance	Private	✓	
	_distributeCommission	Private	✓	nonReentrant

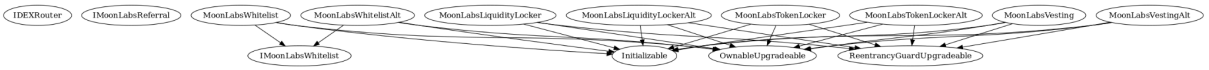
	_handleBurns	Private	✓	
	_calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarnedUSD	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsWhitelist	Implementation	Initializable, IMoonLabsWhitelist, OwnableUpgradeable		
	initialize	Public	✓	initializer
	purchaseWhitelistMLAB	External	✓	-
	purchaseWhitelist	External	✓	-
	purchaseWhitelistWithCode	External	✓	-
	ownerWhitelistAdd	External	✓	onlyOwner
	removeWhitelist	External	✓	onlyOwner
	addPairBlacklist	External	✓	onlyOwner
	removePairBlacklist	External	✓	onlyOwner
	setCostUSD	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner

	setFeeCollector	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setRouter	External	✓	onlyOwner
	setUSDContract	External	✓	onlyOwner
	setMlabToken	External	✓	onlyOwner
	setMlabDiscountPercent	External	✓	onlyOwner
	claimETH	External	✓	onlyOwner
	claimUSD	External	✓	onlyOwner
	getTokenToWhitelist	External		-
	getPairToBlacklist	External		-
	getIsWhitelisted	Public		-
	getMLABFee	Public		-
	_buyWithMLAB	Private	✓	
	_distributeCommission	Private	✓	
	_checkV2Pair	Private		
	_checkV3Pool	Private		
	_checkIfValidPair	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarnedUSD	External	✓	-

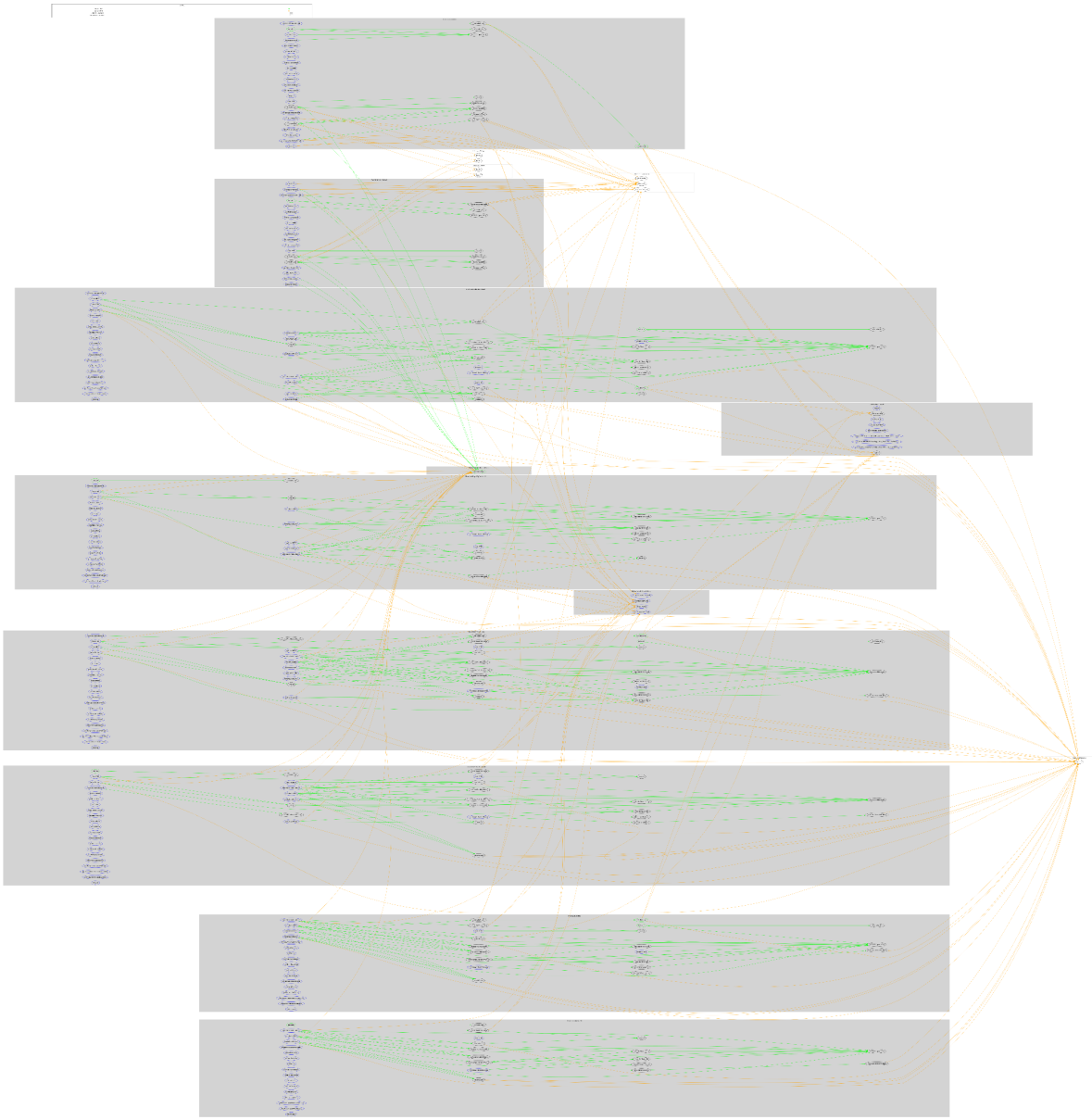
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsWhitelistAlt	Implementation	Initializable, IMoonLabsWhitelist, OwnableUpgradeable		
	initialize	Public	✓	initializer
	purchaseWhitelist	External	✓	-
	purchaseWhitelistWithCode	External	✓	-
	ownerWhitelistAdd	External	✓	onlyOwner
	removeWhitelist	External	✓	onlyOwner
	addPairBlacklist	External	✓	onlyOwner
	removePairBlacklist	External	✓	onlyOwner
	setCostUSD	External	✓	onlyOwner
	setCodeDiscount	External	✓	onlyOwner
	setFeeCollector	External	✓	onlyOwner
	setCodeCommission	External	✓	onlyOwner
	setReferralContract	External	✓	onlyOwner
	setUSDContract	External	✓	onlyOwner
	claimETH	External	✓	onlyOwner
	claimUSD	External	✓	onlyOwner
	getTokenToWhitelist	External		-
	getPairToBlacklist	External		-

	getIsWhitelisted	Public		-
	_distributeCommission	Private	✓	
	_checkV2Pair	Private		
	_checkV3Pool	Private		
	_checkIfValidPair	Private		

Inheritance Graph



Flow Graph



Summary

Moonlab's contracts implement a utility, financial, and locker mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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