



Cyberscope

Audit Report

Moonlabs

March 2023

SHA256

e3580037e1fe6c148ee9090c746c093ba0a832ee2e17b9c7d0da377150f53307
b7b61dc5121be75e61487ac9d6e554c74c4ba7e268fe96998e484d1cec763b99
8f897febcbfe9a904beb778789a70d9fb4d1ad994ba81c9fbcef0020139b61d74
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

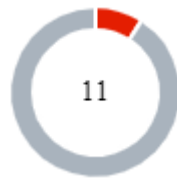
Source Files

Filename	SHA256
IDEXRouter.sol	f6af1340f54f9d239f19282c0177481296b1 b10699e094523136f381714afb30
MoonLabsLiquidityLocker.sol	e3580037e1fe6c148ee9090c746c093ba0 a832ee2e17b9c7d0da377150f53307
MoonLabsLiquidityLockerAlt.sol	f4713e22d679da182da931033210140f656 4bb64a9db52bf30fe8ffea507e619
MoonLabsTokenLocker.sol	b7b61dc5121be75e61487ac9d6e554c74c 4ba7e268fe96998e484d1cec763b99
MoonLabsTokenLockerAlt.sol	6b6cba5974befb58457348f6710d2a6a39f 4bd65970d59e4c19ab33ee8b1ca7c
MoonLabsVesting.sol	8f897febcbfe9a904beb778789a70d9fb4d1 ad994ba81c9fbcef0020139b61d74
MoonLabsVestingAlt.sol	fa6e7d5f1afe7b76f7280b70af90b1618f2fe 186368f3724d411124a2f57f385
MoonLabsWhitelist.sol	dd4187851d23276f0580bc14bc02657614 1b7de603fb665c61f6ed629c92ea54
MoonLabsWhitelistAlt.sol	936a3727b62aaf96c10df43bb6db789d70 0e02e1a24a4dc1ad9fbaab7529f53e

Testing Deploy

Contract Name	Explorer
MoonLabsLiquidity Locker	https://testnet.bscscan.com/address/0x5a69C4666069d8142Ce4C90b0035c9956413b409
MoonLabsReferral	https://testnet.snowtrace.io/address/0xdCF65098B0873153A86C2408dFD012c4847fF299
MoonLabsTokenLo cker	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.swapExactETHForTokensSupportingFeeOnTransferTokens(
            value: _burnMeter )(0, path,
            0x0000000000000000000000000000000000000000dead, 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. 0

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, OwnableUpgradable, ReentrancyGuardUpgradable		
	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		
IMoonLabsReference	Interface			

	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarned	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsTokenLockerAlt	Implementation	Initializable, OwnableUpgradable, ReentrancyGuardUpgradable		
	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, OwnableUpgradable, ReentrancyGuardUpgradable		
	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, OwnableUpgradable, ReentrancyGuardUpgradable		
	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

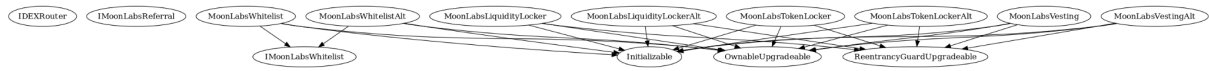
	_handleBurns	Private	✓	
	_calculateLinearWithdraw	Private		
IMoonLabsReferral	Interface			
	checkIfActive	External		-
	getAddressByCode	External		-
	addRewardsEarnedUSD	External	✓	-
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsWhitelist	Implementation	Initializable, IMoonLabsWhitelist, OwnableUpgradable		
	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
	setUSDCContract	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	✓	-

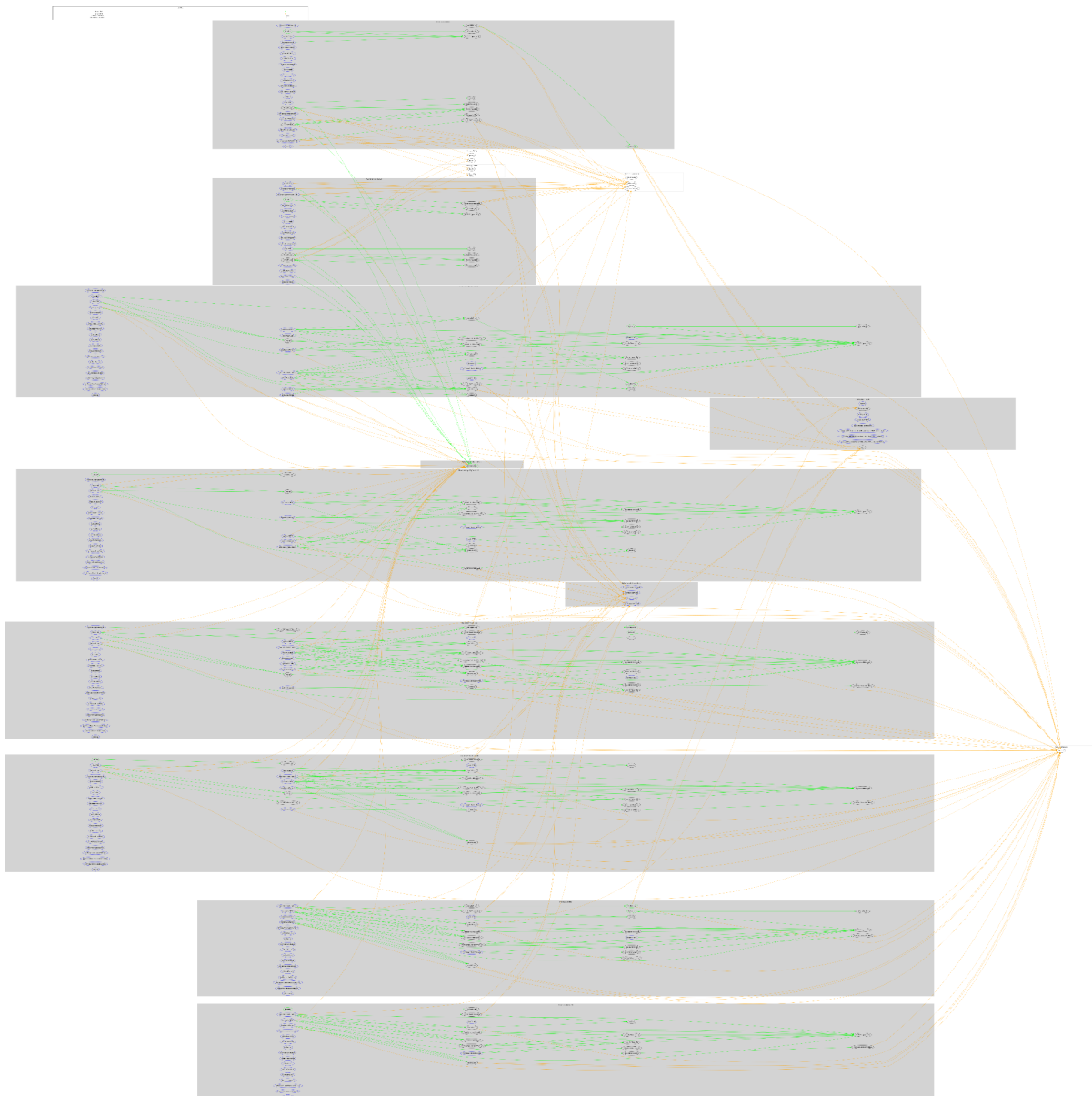
IMoonLabsWhitelist	Interface			
	getIsWhitelisted	External		-
MoonLabsWhitelistAlt	Implementation	Initializable, IMoonLabs Whitelist, OwnableUpgradable		
	initialize	Public	✓	initializer
	purchaseWhitelist	External	✓	-
	purchaseWhitelistWithCode	External	✓	-
	ownerWhitelistAdd	External	✓	onlyOwner
	removeWhitelist	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>