Cyberscope

Audit Report

Circle Launchpad Airdrop

December 2022

Github	https://github.com/monkey-shanti/Circle-Launchpad
Commit	831864399fdc88aaf191f8594ca0d22d09080652
Audited by	© cyberscope

Table of Contents

Contract Review

Contract Name	Testing Deploy
AirdropMaster	https://testnet.bscscan.com/address/0x2934606833FC031ccC75a17E383Eb82CE33ba28d
AirdropManager	https://testnet.bscscan.com/address/0xbfad008F96ee89eFAA1E8755bb07f304f6690E7E
AirdropFactory	https://testnet.bscscan.com/address/0x94d9F7d4Aee923d619FBBF250C5F53E879ad2D18

Audit Updates

Initial Audit	20 Dec 2022

Source Files

Filename	SHA256
AirdropFactory.sol	a1ca82062a468a8bca8b3554811ba6575d6d8dcc8d3ca2f8970dfedb7c60d2dc
AirdropMain.sol	73e3b13b842d1b48dcd556ececd79084a44a37cc93d142b52de84ea7bbb205b9
AirdropManager.sol	b6cd15e497b0856b8a30ba88e55c8606da4f4b9f36ca4cc2478b91b5c0a8464a
multisender/libraries/AddressLib.sol	773d033dd9c33b9799bafc89ab4e7b3693446e0551727b580991292fc8c69add
multisender/libraries/TransferHelper.sol	b337ccfc0cc7ea731f176202350c915cdf77c7aff6f75a893d418918455e88a7
multisender/MultiSender.sol	3c428c5faafa5e80fbc2a4f7fea56cd62b34e093babdf39d5a0743037c7aa850

Introduction

The Circle launchpad Airdrop contract implements a locker mechanism. It consists of a factory, a manager, and the master airdrop contract.

Airdrop Factory

The Airdrop Factory is responsible for creating new airdrops.

Roles

The contract has two roles.

Owner Role

The owner role has the authority to

- setMasterAddress
- setAdminWallet
- setPartnerFee
- setVersion
- setPoolOwner
- setPresalePoolPrice
- setPoolManager
- bnbLiquidity
- poolEmergencyWithdrawToken
- poolEmergencyWithdraw
- poolSetGovernance

User Role

The user has the authority to createSale.

Airdrop Manager

The Airdrop Manager is responsible for adding or removing factories. Additionally, it is responsible for monitoring airdrop factories and keeping registries about them.

Roles

The contract has three roles.

Owner Role

The Owner has the authority to

- addAdminPoolFactory
- addPoolFactories
- removePoolFactory
- bnbLiquidity

AllowedFactory Role

The Allowed Factories have the authority to

- addPoolFactory
- registerPool
- increaseTotalValueLocked
- decreaseTotalValueLocked
- recordContribution
- removePoolForToken

User Role

The users have the authority to

- view isPoolFactory
- view isPoolGenerated
- getPoolsOfLength
- getPoolsForTokenLength
- getPoolsOf
- getPoolsForToken
- getAllPools
- getPoolAt

- getTotalNumberOfPools
- getTotalNumberOfContributedPools
- getAllContributedPools
- getContributedPoolAtIndex
- getTotalNumberOfPools
- getPoolAt
- getCumulativePoolInfo
- getUserContributedPoolInfo

Airdrop Master

The Airdrop Master implements the core functionality of the airdrop.

Airdrop State

The Airdrop has 3 states

- inUse
- completed
- cancelled

Roles

The contract has 5 roles.

Owner Role

The Owner has the authority to

- emergencyWithdrawToken
- emergencyWithdraw
- setGovernance

Whitelisted Role

The Whitelisted role has the authority to

- claim

Operator Role

The Operator has the authority to

- initializeVesting
- addWhitelistedUsers
- addWhitelistedUser
- removeWhitelistedUsers
- isUserWhitelisted
- changeStartAt
- updatePoolDetails

Governance Role

The Governance role is not utilized on the contract implementation.

User Role

The users have the authority to

- getPoolInfo
- getNumberOfWhitelistedUsers
- getWhitelistedUsers
- getUpdatedState
- view userAvalibleClaim

Contract Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
⚪	ICN	Inappropriate Contract Naming	Unresolved
🔴	AFI	Airdrop Finalization Issue	Unresolved
⚪	RCS	Redundant Code Segment	Unresolved
⚪	RDS	Redundant Data Structure	Unresolved
⚪	MC	Missing Check	Unresolved
⚪	L02	State Variables could be Declared Constant	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L05	Unused State Variable	Unresolved
⚪	L06	Missing Events Access Control	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L08	Tautology or Contradiction	Unresolved
⚪	L15	Local Scope Variable Shadowing	Unresolved
⚪	L16	Validate Variable Setters	Unresolved

●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

AFI - Airdrop Finalization Issue

Criticality	Critical
Location	AirdropMain.sol#L291
Status	Unresolved

Description

The Airdrop finalizations will never take place. Due to the unreachable code segment `poolState = PoolState.completed;`.

Since the variable `totalCost` will always be greater than the `totalClaimed`. `totalCost > totalClaimed.`, then the following code segment `poolState = PoolState.completed;` will never be reached. As a result, the following calculation `totalCost - totalClaimed` will never yield a zero value.

```
if(totalCost > totalClaimed) {
    if((totalCost - totalClaimed) == 0) {
        poolState = PoolState.completed;
    }
}
```

Recommendation

The team is advised to carefully check if the implementation follows the expected business logic.

ICN - Inappropriate Contract Naming

Criticality	Minor / Informative
Location	AirdropMain.sol#L48 AirdropManager.sol#L15 AirdropFactory.sol#L41
Status	Unresolved

Description

The Airdrop ecosystem implements a Locker mechanism. Hence, the contract naming is inappropriate.

```
contract AirdropMaster

contract AirdropManager

contract AirdropFactory
```

Recommendation

The team is advised to carefully check if the implementation follows the expected business logic and rename the contracts accordingly.

RCS - Redundant Code Segment

Criticality	Minor / Informative
Location	AirdropMain.sol#L174,189,201,214
Status	Unresolved

Description

The contract is processing the variables `totalCost` and `prevTotalCost` in the same manner, regardless of the function logic. For instance,

1. If the operator adds a whitelisted user, the `totalCost` will always be greater than the `prevTotalCost`.

```
if(totalCost > prevTotalCost) { //true
    _safeTransferFromEnsureExactAmount(token, governance,
address(this), (totalCost - prevTotalCost));
}
```

2. If the operator removes a whitelisted user the `prevTotalCost` will always be greater than the `totalCost`.

```
if(prevTotalCost > totalCost) { //true
    _transferFromEnsureExactAmount(token, governance, prevTotalCost -
totalCost);
}
```

As a result the following code segment could be optimized.

```
if(totalCost > prevTotalCost) {
    _safeTransferFromEnsureExactAmount(token, governance,
address(this), (totalCost - prevTotalCost));
}
if(prevTotalCost > totalCost) {
    _transferFromEnsureExactAmount(token, governance, prevTotalCost -
totalCost);
}
```

Recommendation

The team is advised to remove redundant code segments from the corresponding functions.

RDS - Redundant Data Structure

Criticality	Minor / Informative
Location	AirdropMain.sol#L87,88
Status	Unresolved

Description

The contract utilizes two data structures with almost the same information. The contract is using one mapping to keep a registry of all whitelisted users. And an additional EnumerableSet which keeps a registry of only the whitelisted addresses. In order to monitor the number of whitelisted users. These code segments could be optimized. A segment may be optimized so that it becomes smaller, consumes less memory, executes more rapidly, or performs fewer operations.

```
EnumerableSet.AddressSet private whitelistedUsers;
mapping(address => bool) private isWhitelisted;
```

Recommendation

The team is advised keep a single state variable in order to store the whitelisted users. The enumerable set could be used since it cover all the business requirements. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

MC - Missing Check

Criticality	Minor / Informative
Location	AirdropMain.sol#L128,175,189,201,214 AirdropFactory.sol#L56,156 AirdropManager.sol#L67,71,75,93
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 initializer arguments `_linkAddress[0]` and `_linkAddress[2]` have not been properly sanitized.

```
function initialize(
    address[3] memory _addrs,
    uint256[1] memory _saleInfo,
    string memory _poolDetails,
    address[3] memory _linkAddress,
    uint8 _version
) external override initializer {
    MAX_ALLOCATIONS = 500;
    require(factory == address(0), "Pool: Forbidden");
    require(_addrs[0] != address(0), "Invalid Token address");
    require(_saleInfo[0] >= block.timestamp, "Start time should be in
the future");

    ...
}
```

The `user` and `users` arguments are not properly sanitized.

```
function addWhitelistedUsers(address[] memory users, uint256[] memory
_allocations) external

function addWhitelistedUser(address user, uint256 _allocation) external

function removeWhitelistedUsers(address[] memory users) external

function removeWhitelistedUser(address user) external
```

The `masterPrice` argument is not properly sanitized.

```
function initialize(
    address _master,
    address _poolmanager,
    uint8 _version,
    uint256 _masterPrice,
    bool _IsEnabled
) external initializer {
    __Ownable_init();
    master = _master;
    poolManager = _poolmanager;
    masterPrice = _masterPrice;
    version = _version;
    IsEnabled = _IsEnabled;
}
```

The arguments of `initalizeClone` function are not properly sanitized.

```
function initalizeClone(
    address _pair,
    address[3] memory _addrs,
    uint256[1] memory _saleInfo,
    string memory _poolDetails,
    uint256[3] memory _vestingInit
) internal {
    IAirdrop(_pair).initialize(
        _addrs,
        _saleInfo,
        _poolDetails,
        [poolOwner, poolManager , adminWallet],
        version
    );
    IAirdrop(_pair).initializeVesting(_vestingInit);
}
```

The `price` argument is not properly sanitized.

```
function setPresalePoolPrice(uint256 _price) public onlyOwner {
    masterPrice = _price;
}
```

The `factory` and `factories` arguments are not properly sanitized.

```
function addPoolFactory(address factory) public override
onlyAllowedFactory {
    poolFactories.add(factory);
}

function addAdminPoolFactory(address factory) public onlyOwner {
    poolFactories.add(factory);
}

function addPoolFactories(address[] memory factories) external
onlyOwner {
    for (uint256 i = 0; i < factories.length; i++) {
        addPoolFactory(factories[i]);
    }
}
```

The `registerPool` arguments are not properly sanitized.

```
function registerPool(
    address pool,
    address token,
    address owner,
    uint8 version
) external override onlyAllowedFactory {
    _pools.add(pool);
    _poolsForVersion[version].add(pool);
    _poolsOf[owner].add(pool);
    _poolForToken[token].add(pool);
    emit PoolForTokenCreated(token, pool);
}
```

Recommendation

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

- The variables `_linkAddress[0]` and `_linkAddress[2]` should not be set to zero address.
- The variables `user` and `users` should not be set to zero address.
- The variable `masterPrice` should be greater than zero.
- The `_pair` and `_addrs` addresses should not be set to zero address.
- The `factory` and `factories` address should not be set to zero.
- The address `token` and `owner` should not be set to zero.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	AirdropMain.sol#L78
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 decreases gas consumption of the corresponding transaction.

```
uint256 private tvl
```

Recommendation

Constant state variables can be useful when you want 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 advices to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	AirdropManager.sol#L50,325 AirdropMain.sol#L53,55,129,130,131,132,133,155,174,189,232,360,397 AirdropFactory.sol#L49,57,58,59,60,61,73,78,83,87,92,93,94,95,96,111,112,113,114,151,156,160,165,201
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 your Solidity code, making it easier for others to understand and work with.

The followings are 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 your code.
7. Keep lines short (around 120 characters) to improve readability.

```
event sender(address sender);
address payable _reciever
uint256 _amount
uint256 public MAX_ALLOCATIONS = 500
uint8 public VERSION
address[3] memory _addrs
uint256[1] memory _saleInfo
string memory _poolDetails
address[3] memory _linkAddress
uint8 _version
uint256[3] memory _vestingInit
uint256[] memory _allocations
uint256 _allocation
uint256 _startTime

...
```

Recommendation

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

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

L05 - Unused State Variable

Criticality	Minor / Informative
Location	AirdropManager.sol#L15 AirdropMain.sol#L48,78 AirdropFactory.sol#L41
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used. Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
contract AirdropManager is OwnableUpgradeable, IAirdropManager {
    using EnumerableSetUpgradeable for
EnumerableSetUpgradeable.AddressSet;
    using SafeMathUpgradeable for uint256;
    using SafeERC20Upgradeable for IERC20Upgradeable;

    struct CumulativeLockInfo {
...
        address payaddress,
        address tokenAddress,
        uint256 tokens
    ) public onlyOwner {
        IERC20Upgradeable(tokenAddress).transfer(payaddress, tokens);
    }
}

...
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	AirdropMain.sol#L389
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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
governance = governance_
```

Recommendation

To avoid this issue, 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. By including all required events in the contract and thoroughly testing the contract's functionality, you can help to ensure that the contract performs as intended and does not have any missing events that could cause issues.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	AirdropMain.sol#L168 AirdropFactory.sol#L88,157
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.

```
tgeBps = _vestingInit[0]
version = _version
masterPrice = _price
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, you can help to ensure that the contract 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	AirdropMain.sol#L159,160,161
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 your code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in your Solidity code.

```
require(_vestingInit[1] >= 0, "Invalid cycle")
require(_vestingInit[0] >= 0 && _vestingInit[0] < 10_000, "Invalid bips
for TGE")
require(_vestingInit[2] >= 0 && _vestingInit[2] < 10_000, "Invalid bips
for cycle")
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	AirdropMain.sol#L448,467
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address token
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	AirdropManager.sol#L329 AirdropMain.sol#L389 AirdropFactory.sol#L64,65,169
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.

```
_reciever.transfer(_amount)
governance = governance_
master = _master
poolManager = _poolmanager
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	AirdropManager.sol#L2,3 AirdropMain.sol#L2 AirdropFactory.sol#L2
Status	Unresolved

Description

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

```
pragma solidity ^0.8.4;

pragma solidity ^0.8.4;
pragma experimental ABIEncoderV2;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in. By including all required compiler options and flags in a single pragma directive, you can avoid conflicts and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	AirdropManager.sol#L2 AirdropMain.sol#L2 AirdropFactory.sol#L2
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 you to specify a minimum version of the Solidity compiler that must be used to compile your contract code. This is useful because it allows you to ensure that your contract will be compiled using a version of the compiler that is known to be compatible with your code.

```
pragma solidity ^0.8.4;
```

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	AirdropManager.sol#L337 AirdropMain.sol#L474 AirdropFactory.sol#L177
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.

```
IERC20Upgradeable(tokenAddress).transfer(payaddress, tokens)
IERC20(token).transfer(recipient, 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.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-

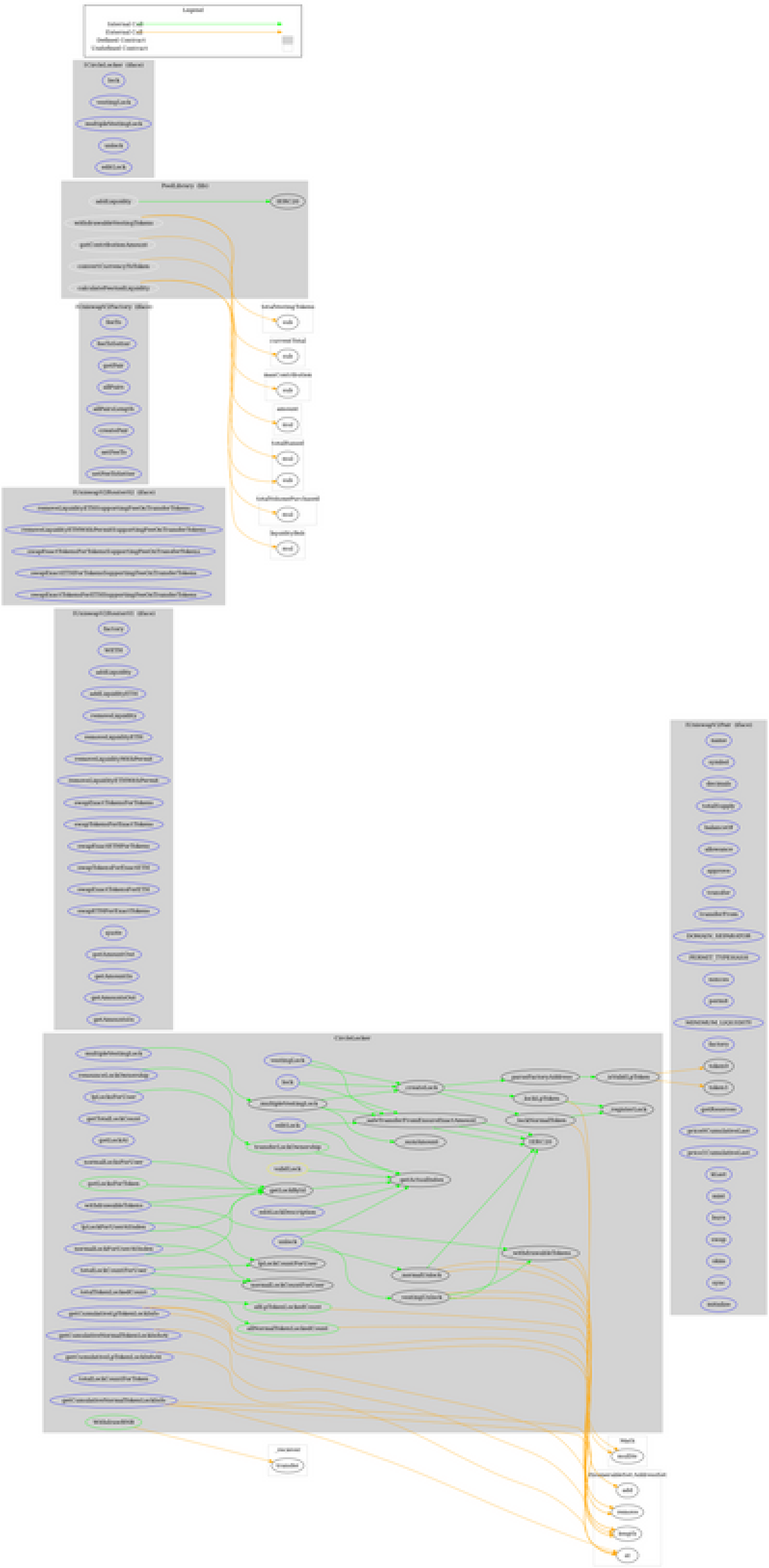
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		

	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
PoolLibrary	Library			
	withdrawableVestingTokens	Internal		
	getContributionAmount	Internal		
	convertCurrencyToToken	Internal		
	addLiquidity	Internal	✓	
	calculateFeeAndLiquidity	Internal		
ICircleLocker	Interface			
	lock	External	✓	-
	vestingLock	External	✓	-
	multipleVestingLock	External	✓	-

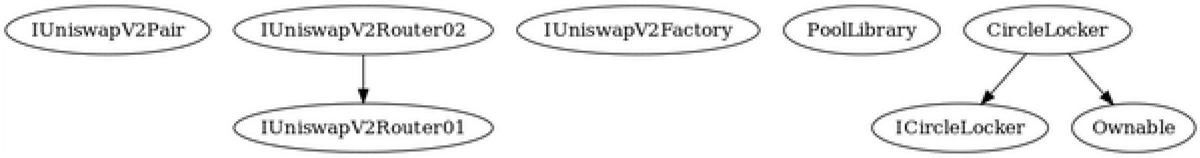
	unlock	External	✓	-
	editLock	External	✓	-
CircleLocker	Implementation	ICircleLocker, Ownable		
	lock	External	✓	-
	vestingLock	External	✓	-
	multipleVestingLock	External	✓	-
	_multipleVestingLock	Internal	✓	
	_sumAmount	Internal		
	_createLock	Internal	✓	
	_lockLpToken	Private	✓	
	_lockNormalToken	Private	✓	
	_registerLock	Private	✓	
	unlock	External	✓	validLock
	_normalUnlock	Internal	✓	
	_vestingUnlock	Internal	✓	
	withdrawableTokens	External		-
	_withdrawableTokens	Internal		
	editLock	External	✓	validLock
	editLockDescription	External	✓	validLock
	transferLockOwnership	Public	✓	validLock
	renounceLockOwnership	External	✓	-
	_safeTransferFromEnsureExactAmount	Internal	✓	
	getTotalLockCount	External		-
	getLockAt	External		-
	getLockById	Public		-
	allLpTokenLockedCount	Public		-
	allNormalTokenLockedCount	Public		-
	getCumulativeLpTokenLockInfoAt	External		-

	getCumulativeNormalTokenLockInfoAt	External		-
	getCumulativeLpTokenLockInfo	External		-
	getCumulativeNormalTokenLockInfo	External		-
	totalTokenLockedCount	External		-
	lpLockCountForUser	Public		-
	lpLocksForUser	External		-
	lpLockForUserAtIndex	External		-
	normalLockCountForUser	Public		-
	normalLocksForUser	External		-
	normalLockForUserAtIndex	External		-
	totalLockCountForUser	External		-
	totalLockCountForToken	External		-
	getLocksForToken	Public		-
	_getActualIndex	Internal		
	_parseFactoryAddress	Internal		
	_isValidLpToken	Private		
	WithdrawBNB	Public	✓	onlyOwner

Contract Flow



Inheritance Graph



Summary

The Airdrop ecosystem contracts implement a 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