Cyberscope

Audit Report

Circle Launchpad Pool

March 2022

Github	https://github.com/monkey-shanti/Circle-Launchpad
Commit	7f6f46693c2710c2dfc986618b6531561f0ddabb
Audited by	© cyberscope

Table of Contents

Contract Review

Repository	https://github.com/monkey-shanti/Circle-Launchpad
Commit	7f6f46693c2710c2dfc986618b6531561f0ddabb

Contract Name	Testing Deploy
CirclePoolManager	https://testnet.bscscan.com/address/0xc49D12328030A2dEFb512c00fc8Fa18D3e51A156
CirclePoolFactory	https://testnet.bscscan.com/address/0x1AA130b41Be8289DaAEbE434D9d9B8C828164F19
CircleFairPool	https://testnet.bscscan.com/address/0x059DBbf08717c231A61312B8706F17d11F701F17
CirclePrivatePool	https://testnet.bscscan.com/address/0xA9B8B8352cD64F4b729759811532FD1A2902DfF1
CirclePresalePool	https://testnet.bscscan.com/address/0xe6A3d2565cfA2c61F0d8a808Da64d16637710822

Audit Updates

Initial Audit	20 Dec 2022 https://github.com/cyberscope-io/audits/blob/main/circleLaunchpad/v1/pool.pdf
Corrected Phase 2	02 Jan 2023 https://github.com/cyberscope-io/audits/blob/main/circleLaunchpad/v2/pool.pdf
Corrected Phase 3	16 Mar 2023

Source Files

Filename	SHA256
FairPool.sol	b71d1fe2e6310996a272812e97eb0bbd751e53d278b3178cf58e25cdb8b4fd7d
interfaces/IERC20Info.sol	1ca3d43543f838b7014dc2021dc5338492249523f0d9038eff73befa3f15ed1b
interfaces/IFairPool.sol	2bf91078f72b43b4775e8288dd81ab57c79d11b142b068891fd679630220e9a2
interfaces/IPool.sol	7ecfe46b3c75d5c14fb4b1041bdcf7dbb538c06a926dcc642d2580234bc50941
interfaces/IPoolFactory.sol	494c0dbb60f69829c5e5ce9fc1d6601576766d4d27a0f8e5d1a3da29dee7c017
interfaces/IPoolManager.sol	48a5288028d0d4dd2861b9b7a539c8dc0241a46fba7783addf049c0e0e6f9d1d
interfaces/IPrivatePool.sol	7d0a66447b55878209e03717a70e424c4e7149dd35f610920ae3dc7bf4c2ca6b
interfaces/IUniswapV2Pair.sol	5631411f67c8741031e9bfdd27fad3c81554c0c92a37dce6990b618ef634cd0a
interfaces/PoolLibrary.sol	f209394b1e0c66187d6e6f86f5de7708930fe8f282c3fbfbe69e507dc5133939
libraries/LibEnsureSafeTransfer.sol	e298b11772c723015c7dedcd8d62d8228ee29ffcaad8c794e3e865c3765c53dd
libraries/LibPresale.sol	af71027b0d7f1e8ca1a81c8c5fa77ca6d71e665ea8cfbb63875a94750384e962
libraries/LibTier.sol	4aedc295fb6e752b1beea948f23aac1f29f885eaa9891df7388c59d2110a4d3b
PoolFactory.sol	ba6ce1af75d8900432a484bbb90bce725895008db4f7ee877f60672d036f0dd7
PoolManager.sol	24f8923258cc7a17686fc3b329a3a217c62739d1ee9c0d60153fe9981be8a068
PresalePool.sol	f176246c160946d3b3e571fdd420564bc62d5466be09fff192ae92247c435c31

PrivatePool.sol	71b760c6e6f6f0c0790c7c861ba9e6a57b85047a2532e833dd15e4208c909363
utils/Utility.sol	bb982ca156ddbd0ea26ba803843a755ff4ad6440addadc577d3ba8335f8e0705

Introduction

The Circle launchpad pool implements a presale mechanism. It consists of a factory, a manager, a fair pool, a presale pool, and a private pool contract.

CirclePoolFactory

The Circle PoolFactory is responsible for creating new presales.

Roles

The contract has two roles.

Owner Role

The owner role has the authority to

- setMasterAddress
- setFairAddress
- setPrivateAddress
- setAdminWallet
- setPartnerFee
- setVersion
- setFees
- setPaymentCurrency
- setPoolOwner
- setkycPrice
- setAuditPrice
- setPresalePoolPrice
- setPrivatePoolPrice
- setFairPoolPrice
- setPoolManager
- bnbLiquidity
- transferAnyERC20Token
- poolEmergencyWithdrawLiquidity

- poolEmergencyWithdrawToken
- poolEmergencyWithdraw
- poolSetGovernance

User Role

The user has the authority to

- getFee
- getPaymentCurrency
- createSale
- createPrivateSale
- createFairSale

CirclePoolManager

The Circle Pool Manager is responsible for adding or removing the presale. Additionally, it's responsible for monitoring pool factories and keeping registries about them.

Roles

The contract has three roles.

Owner Role

The owner role has the authority to

- addAdminPoolFactory
- addPoolFactories
- removePoolFactory
- initializeTopPools
- bnbLiquidity
- transferAnyERC20Token

AllowedFactory Role

The Operator has the authority to

- addPoolFactory
- registerPool
- registerPrivatePool
- increaseTotalValueLocked
- decreaseTotalValueLocked
- recordContribution
- removePoolForToken
- removePrivatePoolForToken
- addTopPool
- removeTopPool

User Role

The user has the authority to

- isPoolGenerated

- poolForToken
- privatePoolForToken
- getPoolsOf
- getAllPools
- getPoolAt
- getTotalNumberOfContributedPools
- getAllContributedPools
- getContributedPoolAtIndex
- getTotalNumberOfPools
- getPoolAt
- getTopPool
- getCumulativePoolInfo

CircleFairPool

The Circle FairPool implements a fair launch mechanism.

Pool States

The pool has 2 states.

- inProgress
- notInProgress

Roles

The contract has two roles.

Owner Role

The owner role has the authority to

- emergencyWithdrawLiquidity
- emergencyWithdrawToken
- emergencyWithdraw
- updateCompletedKyc
- setGovernance

Operator Role

The Operator has the authority to

- finalize
- cancel
- withdrawLeftovers
- withdrawLiquidity
- updatePoolDetails

Governance Role

The Governance role is not utilized on the contract implementation.

User Role

The user has the authority to

- contribute
- claim

- withdrawContribution
- getPrice
- getPoolInfo
- convert
- View userAvalibleClaim

CirclePrivatePool

The Circle PrivatePool implements a private presale.

Pool States

The pool has 3 states.

- inUse
- completed
- cancelled

Roles

The contract has two roles.

Owner Role

The owner role has the authority to

- emergencyWithdrawToken
- emergencyWithdraw
- setGovernance

Operator Role

The Operator has the authority to

- setWhitelist
- finalize
- cancel
- withdrawLeftovers
- updatePoolDetails
- startPublicSaleNow
- startTier2SaleNow
- changeWhitelist
- changeTierDates

Governance Role

The Governance role is not utilized in the contract implementation.

Whitelisted Role

The Whitelisted role is not utilized in the contract implementation.

User Role

The user has the authority to

- getPoolInfo
- getNumberOfWhitelistedUsers
- getWhitelistedUsers
- contribute
- withdrawContribution
- claim
- getContributionAmount
- remainingContribution
- userAvalibleClaim
- getTier

CirclePresalePool

The Circle Presale rPool implements a regular presale mechanism.

Pool States

The pool has 3 states.

- inUse
- completed
- cancelled

Roles

The contract has two roles.

Owner Role

The owner role has the authority to

- emergencyWithdrawLiquidity
- emergencyWithdrawToken
- emergencyWithdraw
- setGovernance

Operator Role

The Operator has the authority to

- addWhitelistedUsers
- removeWhitelistedUsers
- finalize
- cancel
- withdrawLiquidity
- updatePoolDetails
- startPublicSaleNow
- changeWhitelist
- startTier2SaleNow
- changeTierDates

Governance Role

The Governance role is not utilized on the contract implementation.

User Role

The user has the authority to

- getNumberOfWhitelistedUsers
- getWhitelistedUsers
- getPoolInfo
- contribute
- claim
- withdrawContribution
- getContributionAmount
- remainingContribution
- userAvailableClaim

Contract Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MSC	Missing Sanity Check	Unresolved
●	CR	Code Readability	Unresolved
●	MTV	Missing Token Validation	Unresolved
●	RDS	Redundant Data Structure	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	PoolManager.sol#L88
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 variable `emergencyWithdrawFees` should not be greater that the percentage denominator.

```
function setEmergencyWithdrawFees(
    uint256 _fees
) external onlyOwner validAmount(_fees) {
    emit EmergencyWithdrawFeesChanged(
      address(this),
      emergencyWithdrawFees,
      _fees
    );
    emergencyWithdrawFees = _fees;
}
```

Recommendation

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

CR - Code Readability

Criticality	Minor / Informative
Location	PoolManager.sol#L132,145,339,416
Status	Unresolved

Description

There are code segments that are hard to read. It is unclear for the reader to distinguish which pool state is utilized `poolState[pool] = 1`.

```
_poolState[pool] = 1;
```

Recommendation

The team is advised to utilize descriptive values like an enumaration, which can make the contract easier to read and maintain.

MTV - Missing Token Validation

Criticality	Minor / Informative
Location	PrivatePool.sol#L165
Status	Unresolved

Description

Payment tokens are not validated in Private Pool like the Presale Pool and Fair Pool. The contract is processing payment tokens that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

```
address public paymentToken;
```

Recommendation

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

RDS - Redundant Data Structure

Criticality	Minor / Informative
Location	PresalePool.sol#L96,97 PrivatePool.sol#L36,37
Status	Unresolved

Description

The contract utilizes two data structures with the same information.

- One mapping to keep a registry of the amount that the users contributed.
- One mapping that keeps a registry of the amount that the users purchased.

The contribution amount is proportional to the purchased amount. The rate between these two amounts is fixed. Storing two state variables increases gas consumption and decreases readability.

```
mapping(address => uint256) public contributionOf;
mapping(address => uint256) public purchasedOf;
```

Recommendation

The team is advised to remove the purchasedOf data structure. The information could be reshaped by using the proportional contributionOf value. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PrivatePool.sol#L31,225,226,227,424,721,769,775,776,777,804 PresalePool.sol#L29,30,234,235,236,458,841,852,863,864,865 PoolManager.sol#L73,89,409,410 PoolFactory.sol#L39,75,76,77,78,79,80,81,82,83,110,116,122,128,134,139,143,149,154,177,195,196,275,276,326,327,461,467,474,481,487,493,494,540 FairPool.sol#L25,26,40,149,150,151,209,485
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.

```
uint8 public VERSION
uint256[2] memory _fees
address[3] memory _linkAddress
uint8 _version
uint256 _funds
address _userAddress
bool _whitelist
uint256 _endTime1
uint256 _endTime2
uint256 _publicStartTime
uint256 _tier
uint public MINIMUM_LOCK_DAYS = 30 days
event sender(address sender);
uint256 _fees

...
```

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	PoolFactory.sol#L151
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.

```
feeIndex = _index
```

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	PrivatePool.sol#L248,253,254,258 PresalePool.sol#L277,290,291,295 PoolFactory.sol#L150,155 FairPool.sol#L163
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(
      _fees[1] >= 0 && _fees[1] <= 100 && _fees[0] >= 0 && _fees[0] <=
100,
      "Invalid fee settings. Must be percentage (0 -> 100)"
    )
require(presale.cycle >= 0, "Invalid cycle")

...
    )

require(
      presale.cycleBps >= 0 && presale.cycleBps < 10_000,
      "Invalid bips for cycle"
    )
require(_index >= 0, "Invalid Fee Index")

...
```

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	PoolFactory.sol#L218,295,367
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(
      paymentCurrencies[presale.paymentToken] == true ||
        address(0) == presale.paymentToken,
      "Invalid payment token"
    )
```

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.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	PoolManager.sol#L351
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.

```
string memory poolDetails
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	PrivatePool.sol#L596,737 PresalePool.sol#L558 PoolFactory.sol#L437,442
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.

```
currentTotal =
            (((block.timestamp - tgeDate) / cycle) *
cycleReleaseAmount) +
            tgeReleaseAmount

uint256 currentTotal = (((block.timestamp - tgeDate) / cycle) *
      cycleReleaseAmount) + tgeReleaseAmount

uint256 totalToken = (((_rate * _hardcap) / 10 ** decimals)).add(
      _listingType == 0
        ? (((_hardcap * _Lrate) / 10 ** decimals) * _liquidityPercent)
/ 100
        : 0
    )
```

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	PoolManager.sol#L346,349,350,378,379
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 token
string memory name
string memory symbol
string memory poolDetails
```

Recommendation

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

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CircleFairPool	Implementation	OwnableUpgradeable, IFairPool, ReentrancyGuard, Utility		
	initialize	External	✓	initializer
	getDecimal	Public		-
	contribute	Public	Payable	inProgress nonReentrant
	claim	Public	✓	nonReentrant
	withdrawContribution	External	✓	nonReentrant
	finalize	External	✓	onlyOperator nonReentrant
	getPrice	Public		-
	getPoolInfo	External		-
	cancel	External	✓	onlyOperator
	withdrawLeftovers	External	✓	onlyOperator
	withdrawLiquidity	External	✓	onlyOperator
	emergencyWithdrawLiquidity	External	✓	onlyOwner
	emergencyWithdrawContribution	Public	Payable	inProgress nonReentrant
	emergencyWithdrawToken	External	✓	onlyOwner
	emergencyWithdraw	External	✓	onlyOwner
	updatePoolDetails	External	✓	onlyOperator
	updateCompletedKyc	External	✓	onlyOwner
	setGovernance	External	✓	onlyOwner validAddress
	userAvalibleClaim	Public		-

IERC20Info	Interface			
	decimals	External		-
	name	External		-
	symbol	External		-
	supply	External		-
IFairPool	Interface			
	initialize	External	✓	-
	emergencyWithdrawLiquidity	External	✓	-
	emergencyWithdraw	External	✓	-
	setGovernance	External	✓	-
	emergencyWithdrawToken	External	✓	-
	getPoolInfo	External		-
IPool	Interface			
	initialize	External	✓	-
	setGovernance	External	✓	-
	emergencyWithdrawToken	External	✓	-
	getPoolInfo	External		-
IPoolFactory	Interface			
	increaseTotalValueLocked	External	✓	-
	decreaseTotalValueLocked	External	✓	-
	removePoolForToken	External	✓	-
	recordContribution	External	✓	-
	addTopPool	External	✓	-
	removeTopPool	External	✓	-
	removePrivatePoolForToken	External	✓	-

	getEmergencyWithdrawFees	External		-
IPoolManager	Interface			
	increaseTotalValueLocked	External	✓	-
	decreaseTotalValueLocked	External	✓	-
	removePoolForToken	External	✓	-
	removePrivatePoolForToken	External	✓	-
	recordContribution	External	✓	-
	isPoolGenerated	External		-
	addTopPool	External	✓	-
	removeTopPool	External	✓	-
	registerPool	External	✓	-
	registerPrivatePool	External	✓	-
	poolForToken	External		-
	privatePoolForToken	External		-
	addPoolFactory	External	✓	-
	getEmergencyWithdrawFees	External		-
IPrivatePool	Interface			
	initialize	External	✓	-
	getPoolInfo	External		-
	emergencyWithdraw	External	✓	-
	setGovernance	External	✓	-
	emergencyWithdrawToken	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-

	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	factory	External		-
	token0	External		-
	token1	External		-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
IUniswapV2Router02	Interface	IUniswapV2 Router01		
IUniswapV2Factory	Interface			
	getPair	External		-
PoolLibrary	Library			
	withdrawableVestingTokens	Internal		
	getContributionAmount	Internal		
	convertCurrencyToToken	Internal		
	addLiquidity	Internal	✓	
	calculateFeeAndLiquidity	Internal		

LibEnsureSafeTransfer	Library			
	safeTransferFromEnsureExactAmount	Internal	✓	validAddress validAddress validAddress validAmount
	transferEnsureExactAmount	Internal	✓	validAddress validAddress validAmount
	transferExactNativeOrToken	Internal	✓	
	transferExactNative	Internal	✓	validAddress validAmount
	safeTransferFrom	Internal	✓	validAddress validAddress validAddress validAmount
	safeTransfer	Internal	✓	validAddress validAddress validAmount
	transferNativeOrToken	Internal	✓	
	transferNative	Internal	✓	validAddress validAmount
LibPresale	Library			
LibTier	Library			
CirclePoolFactory	Implementation	OwnableUpgradeable, Utility		
	initialize	External	✓	validAddress validAddress validAddress validAddress initializer
		External	Payable	-
	setIsEnabled	Public	✓	onlyOwner
	setMasterAddress	Public	✓	onlyOwner validAddress

	setFairAddress	Public	✓	onlyOwner validAddress
	setPrivateAddress	Public	✓	onlyOwner validAddress
	setAdminWallet	Public	✓	onlyOwner validAddress
	setPartnerFee	Public	✓	onlyOwner
	setVersion	Public	✓	onlyOwner
	setFeeIndex	Public	✓	onlyOwner
	setFees	Public	✓	onlyOwner
	getFee	Public		-
	setPaymentCurrency	Public	✓	onlyOwner
	getPaymentCurrency	Public		-
	initializeClone	Internal	✓	
	createSale	External	Payable	-
	initializePrivateClone	Internal	✓	
	createPrivateSale	External	Payable	-
	initializeFairClone	Internal	✓	
	createFairSale	External	Payable	-
	checkFees	Internal	✓	
	fairFees	Internal	✓	
	checkPrivateSalefees	Internal	✓	
	_feesCount	Internal		
	_feesFairCount	Internal		
	setPoolOwner	Public	✓	onlyOwner
	setPresalePoolPrice	Public	✓	onlyOwner validAmount
	setPrivatePoolPrice	Public	✓	onlyOwner validAmount
	setFairPoolPrice	Public	✓	onlyOwner validAmount
	setPoolManager	Public	✓	onlyOwner

	bnbLiquidity	Public	✓	onlyOwner validAddress validAmount
	transferAnyERC20Token	Public	✓	onlyOwner
	poolEmergencyWithdrawToken	Public	✓	onlyOwner
	poolSetGovernance	Public	✓	onlyOwner
CirclePoolManager	Implementation	OwnableUpgradeable, IPoolManager, Utility		
		External	Payable	-
	initialize	External	✓	initializer
	setEmergencyWithdrawFees	External	✓	onlyOwner validAmount
	getEmergencyWithdrawFees	External		-
	addPoolFactory	Public	✓	onlyAllowedFactory
	addAdminPoolFactory	Public	✓	onlyOwner
	addPoolFactories	External	✓	onlyOwner
	removePoolFactory	External	✓	onlyOwner
	isPoolGenerated	Public		-
	poolForToken	External		-
	privatePoolForToken	External		-
	registerPool	External	✓	onlyAllowedFactory
	registerPrivatePool	External	✓	onlyAllowedFactory
	increaseTotalValueLocked	External	✓	onlyAllowedFactory
	decreaseTotalValueLocked	External	✓	onlyAllowedFactory
	recordContribution	External	✓	onlyAllowedFactory
	removePoolForToken	External	✓	onlyAllowedFactory
	removePrivatePoolForToken	External	✓	onlyAllowedFactory

	getPoolsOf	Public		-
	getAllPools	Public		-
	getPoolAt	Public		-
	getTotalNumberOfPools	Public		-
	getTotalNumberOfContributedPools	Public		-
	getAllContributedPools	Public		-
	getContributedPoolAtIndex	Public		-
	getTotalNumberOfPools	Public		-
	getPoolAt	Public		-
	getTopPool	Public		-
	initializeTopPools	Public	✓	onlyOwner
	addTopPool	External	✓	onlyAllowedFactory
	removeTopPool	External	✓	onlyAllowedFactory
	getCumulativePoolInfo	External		-
	bnbLiquidity	Public	✓	onlyOwner validAddress validAmount
	transferAnyERC20Token	Public	✓	onlyOwner validAddress validAddress validAmount
CirclePresalePool	Implementation	OwnableUpgradeable, IPool, ReentrancyGuard, Utility		
	initialize	External	✓	initializer
	addWhitelistedUsers	External	✓	-
	removeWhitelistedUsers	External	✓	-
	setWhitelist	Internal	✓	onlyOperator
	getPoolInfo	External		-

	contribute	Public	Payable	inProgress nonReentrant
	claim	Public	✓	nonReentrant
	_withdrawableTokens	Internal		
	withdrawContribution	External	✓	nonReentrant
	finalize	External	✓	onlyOperator nonReentrant
	cancel	External	✓	onlyOperator
	withdrawLeftovers	External	✓	onlyOperator
	withdrawLiquidity	External	✓	onlyOperator
	emergencyWithdrawContribution	Public	Payable	nonReentrant
	_withdrawContribution	Internal	✓	
	emergencyWithdrawToken	External	✓	onlyOwner
	updatePoolDetails	External	✓	onlyOperator
	setGovernance	External	✓	onlyOwner validAddress
	userAvailableClaim	Public		-
	startPublicSaleNow	External	✓	onlyOperator inProgress
	changeWhitelist	External	✓	onlyOperator
	startTier2SaleNow	External	✓	onlyOperator inProgress
	changeTierDates	External	✓	onlyOperator notInProgress
	getPaymentTokenDecimals	Public		-
CirclePrivatePool	Implementation	OwnableUpgradeable, IPrivatePool, ReentrancyGuard, Utility		
	initialize	External	✓	initializer
	addWhitelistedUsers	External	✓	-
	removeWhitelistedUsers	External	✓	-
	setWhitelist	Internal	✓	onlyOperator

	getPoolInfo	External		-
	getNumberOfWhitelistedUsers	Public		-
	getWhitelistedUsers	Public		-
	contribute	Public	Payable	inProgress
	withdrawContribution	External	✓	nonReentrant
	emergencyWithdrawContribution	Public	Payable	nonReentrant
	claim	Public	✓	nonReentrant onlyGovernance
	_withdrawableTokens	Internal		
	finalize	External	✓	onlyOperator nonReentrant
	cancel	External	✓	onlyOperator
	withdrawLeftovers	External	✓	onlyOperator nonReentrant
	emergencyWithdrawToken	External	✓	onlyOwner
	emergencyWithdraw	External	✓	onlyOwner
	updatePoolDetails	External	✓	onlyOperator
	setGovernance	External	✓	onlyOwner validAddress
	getContributionAmount	Public		-
	remainingContribution	Public		-
	userAvalibleClaim	Public		-
	startPublicSaleNow	External	✓	onlyOperator inProgress
	startTier2SaleNow	External	✓	onlyOperator inProgress
	changeWhitelist	External	✓	onlyOperator
	changeTierDates	External	✓	onlyOperator notInProgress
	getTier	Public		-
Utility	Implementation			

Contract Flow

Inheritance Graph

Summary

The Pool ecosystem contracts implement a pool 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