Cyberscope

Audit Report

Circle Launchpad Pool

December 2022

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

Table of Contents

Contract Review

Contract Name	Testing Deploy
CirclePoolManager	https://testnet.bscscan.com/address/0x46623e9AF9b3a416B765792C6dFa54cD074639c7
CirclePoolFactory	https://testnet.bscscan.com/address/0x04c384e331c5b3EB9D902A759a572793fFB21aE1
CircleFairPool	https://testnet.bscscan.com/address/0xA5374b1DD49d9F75475132d568024691530B3f9A
CirclePrivatePool	https://testnet.bscscan.com/address/0x40FF5999CC0C610DcC8c9ec8A5269C011cCC28e3
CirclePresalePool	https://testnet.bscscan.com/address/0xBc792aC820e533088Ca5dec62bf534FF6CD8b351

Audit Updates

Initial Audit	20 Dec 2022

Source Files

Filename	SHA256
FairPool.sol	a332b27f2fa1d783665a5e6bc0ee5e840d9866b5c23eab93946c50ea007c0039
interfaces/IERC20Info.sol	984c82e1bf3eebd02e74f2939688fd8c4e821659b8ad3df1c9cd75da3744dfca
interfaces/IFairPool.sol	2ee884bf506a238835c81b21971daa929ded1023a80e07e3e8005b6d2bdf8f16
interfaces/IPool.sol	2a3281b2cfff7488a7cc4693e1be6629c90d63b6a77e28fabc7fd5a14a64fa13
interfaces/IPoolFactory.sol	a2e8f6a6031f2dd9fcfb3f81510a8e5f50d354bee4a7e868313a11708a1afe94
interfaces/IPoolManager.sol	db451a1d1622f254a30b6c7ed0ce091edd831fd119ae90422bb3f589b8cb26fe
interfaces/IPrivatePool.sol	73df825a5e3881762e07e741d5b708af462133f6d63df1d7a320f449ade179d8
interfaces/IUniswapV2Pair.sol	123cb0b5508420d58ba182bc3218fb9c670efd16de72f33415b0657a93873538
interfaces/PoolLibrary.sol	2d5a552523e29e49a13d68a4c7d81b99541e958b179d5128ab5baaa14bdcda54
libraries/LibPresale.sol	af71027b0d7f1e8ca1a81c8c5fa77ca6d71e665ea8cfbb63875a94750384e962
libraries/LibTier.sol	4aedc295fb6e752b1beea948f23aac1f29f885eaa9891df7388c59d2110a4d3b
PoolFactory.sol	9bf756de5a2e6072211cea852c496f8aef30d218655af68e6791113b02a038a7
PoolManager.sol	f2c7c4139fb5cd7e66897e93febdb8faa89fd60ec59ffde646a77f6fc4600bb2

PresalePool.sol	a655186e668b18e076bd606bfc1b1c675755dd20ebc3c57f22ba2c77c3ce834b
PrivatePool.sol	4670e1baf7b43d4eb7798968914621634bd6547ab71a39e82c3384db026f34fc

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
- getUserContributedPoolInfo

CircleFairPool

The Circle FairPool implements a fair launch mechanism.

Pool States

The pool has 4 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
- getUpdatedState
- 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
- convert
- getUpdatedState
- 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
- liquidityBalance
- remainingContribution
- convert
- getUpdatedState
- userAvailableClaim
- getTier

Contract Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CSFMI	Create Sale Fee Manipulation Issue	Unresolved
●	RDS	Redundant Data Structure	Unresolved
●	MTV	Missing Token Validation	Unresolved
●	CRI	Code Readability Issue	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	CR	Code Repetition	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

●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

CSFMI - Create Sale Fee Manipulation Issue

Criticality	Minor / Informative
Location	PoolFactory.sol#L192
Status	Unresolved

Description

The contract users have the authority to provide the corresponding feeIndex. The user may take advantage of it by calling the `createSale` function with an feeIndex that is out of the fees bounds. As a result, no fee will be applied to the presale.

```
Fee memory fee = fees[presale.feeIndex];
```

Recommendation

The contract could check if the provided fee index is between the boundaries.

RDS - Redundant Data Structure

Criticality	Minor / Informative
Location	FairPool.sol#L103,104 PrivatePool.sol#L70,71 PresalePool.sol#L96,97
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.

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 belong to the supported payment tokens.

```
address public paymentToken;
```

Recommendation

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

CRI - Code Readability Issue

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 enumeration, which can make the contract easier to read and maintain.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	FairPool.sol#L223 PrivatePool.sol#L319 PresalePool.sol#L353
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 they transfer the token 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.

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

```
function contribute(uint256 _funds) public payable inProgress {
    uint256 requiredFunds = _funds;
    if(paymentToken == address(0)) {
        require(msg.value > 0, "Cant contribute 0");
        requiredFunds = msg.value;
    } else {
        IERC20(paymentToken).transferFrom(msg.sender, address(this),
requiredFunds);
    }
    ...
}
```

Recommendation

The team is advised to take into consideration the actual amount that has 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 should take into consideration the potential deducted amount after the transfer.

CR - Code Repetition

Criticality	Minor / Informative
Location	FairPool.sol#L278,316,328,340 PrivatePool.sol#L367,399,470,508 PresalePool.sol#L467,510,522,534
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

The following code segment is repetitive in four functions.

```
if(paymentToken == address(0)) {
    payable(msg.sender).sendValue(refundAmount);
} else {
    IERC20(paymentToken).safeTransfer(msg.sender, refundAmount);
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help to reduce the complexity and size of your contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

MC - Missing Check

Criticality	Minor / Informative
Location	FairPool.sol#L162
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 initialize argument is not properly sanitized.

```
function initialize(
    LibPresale.FairLaunch memory presale,
    uint256[2]  memory _fees, // [0] nativeFee ,[1] = tokenFee
    address[3]  memory _linkAddress, // [0] factory ,[1] = manager
    uint8 _version
) external override  initializer {
...
}
```

Recommendation

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

- The variables `presale.router` should not be set to zero.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	PrivatePool.sol#L68 FairPool.sol#L67,68,69,70,71,72,73
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private tvl
bool public audit
bool public kyc
bool public auditStatus
bool public kycStatus
string public auditLink
string public kycLink
string public ownerMail
```

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	PrivatePool.sol#L24,167,168,169,319,547,608,613,634 PresalePool.sol#L23,24,27,190,191,192,353,688,698,707,727 PoolManager.sol#L65,480 PoolFactory.sol#L39,62,63,64,65,66,67,68,69,70,89,94,99,104,109,113,118,141,156,157,226,227,272,273,447,452,456,460,464,468,472,477,522 FairPool.sol#L23,24,38,164,165,166,223,503
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 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 your 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 _endTime
uint256 _tier
uint public MINIMUM_LOCK_DAYS = 30 days

...
```

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	PrivatePool.sol#L18,68 PresalePool.sol#L17 PoolManager.sol#L19 PoolFactory.sol#L20 FairPool.sol#L17
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 CirclePrivatePool is OwnableUpgradeable,
IPrivatePool,ReentrancyGuard {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    using Address for address payable;
    using EnumerableSet for EnumerableSet.AddressSet;

...
    }


    function getTier(uint256 _tier) public view returns (LibTier.Tier
memory) {
        return _tier == 1 ? tier1 : tier2;
    }
}

...
```

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	PrivatePool.sol#L521 PresalePool.sol#L648 FairPool.sol#L478
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 implements 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	PoolFactory.sol#L114,461,465,469
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.

```
version = _version
masterPrice = _price
privatemasterPrice = _price
fairmasterPrice = _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	PrivatePool.sol#L176,184,185,186,615 PresalePool.sol#L203,215,216,217,709 PoolFactory.sol#L119 FairPool.sol#L177
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(
            _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.tgeBps >= 0 && presale.tgeBps < 10_000, "Invalid bips
for TGE")
require(presale.cycleBps >= 0 && presale.cycleBps < 10_000, "Invalid
bips for cycle")
_endTime >= 0

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	PoolFactory.sol#L423
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _feesPrivateCount(
        uint256 _rate,
        uint256 _hardcap,
        uint256 _fees
    ) internal pure returns (uint256) {
        uint256 totalToken = (((_rate * _hardcap) / 10**18));
        uint256 totalFees = (((((_rate * _hardcap) / 10**18)) * _fees)
/ 100);
        uint256 total = totalToken.add(totalFees);
        return total;
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	PoolFactory.sol#L183
Status	Unresolved

Description

The 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#L342,343,344,346,419,420,421,423
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 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.

```
uint8[] memory saleType
uint256[] memory info
string memory name
string memory poolDetails
```

Recommendation

By declaring local variables before using them, you can ensure that your contract 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	PoolManager.sol#L296,373,374,450,451
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in your contract. It's important to always initialize local variables with appropriate values before using them.

```
TopPoolInfo memory tmp
string memory name
string memory poolDetails
```

Recommendation

By initializing local variables before using them, you can help ensure that your contract functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	PrivatePool.sol#L521 PresalePool.sol#L648 PoolManager.sol#L484 PoolFactory.sol#L73,74,75,76,481 FairPool.sol#L478
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.

```
governance = governance_
_reciever.transfer(_amount)
master = _master
privatemaster = _privatemaster
poolManager = _poolmanager
fairmaster = _fairmaster
```

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	PrivatePool.sol#L2 PresalePool.sol#L2 PoolManager.sol#L2,3 PoolFactory.sol#L2 interfaces/IPrivatePool.sol#L1 interfaces/IPoolManager.sol#L1 interfaces/IPoolFactory.sol#L1 interfaces/IFairPool.sol#L1 FairPool.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	PrivatePool.sol#L2 PresalePool.sol#L2 PoolManager.sol#L2 PoolFactory.sol#L2 interfaces/IPrivatePool.sol#L1 interfaces/IPoolManager.sol#L1 interfaces/IPoolFactory.sol#L1 interfaces/IFairPool.sol#L1 FairPool.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	PrivatePool.sol#L325,503 PresalePool.sol#L359 PoolManager.sol#L492 PoolFactory.sol#L489 FairPool.sol#L229
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.

```
IERC20(paymentToken).transferFrom(msg.sender, address(this),
requiredFunds)
IERC20(tokenAddress).transfer(payaddress, tokens)
IERC20Upgradeable(tokenAddress).transfer(payaddress, tokens)
```

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	✓	-

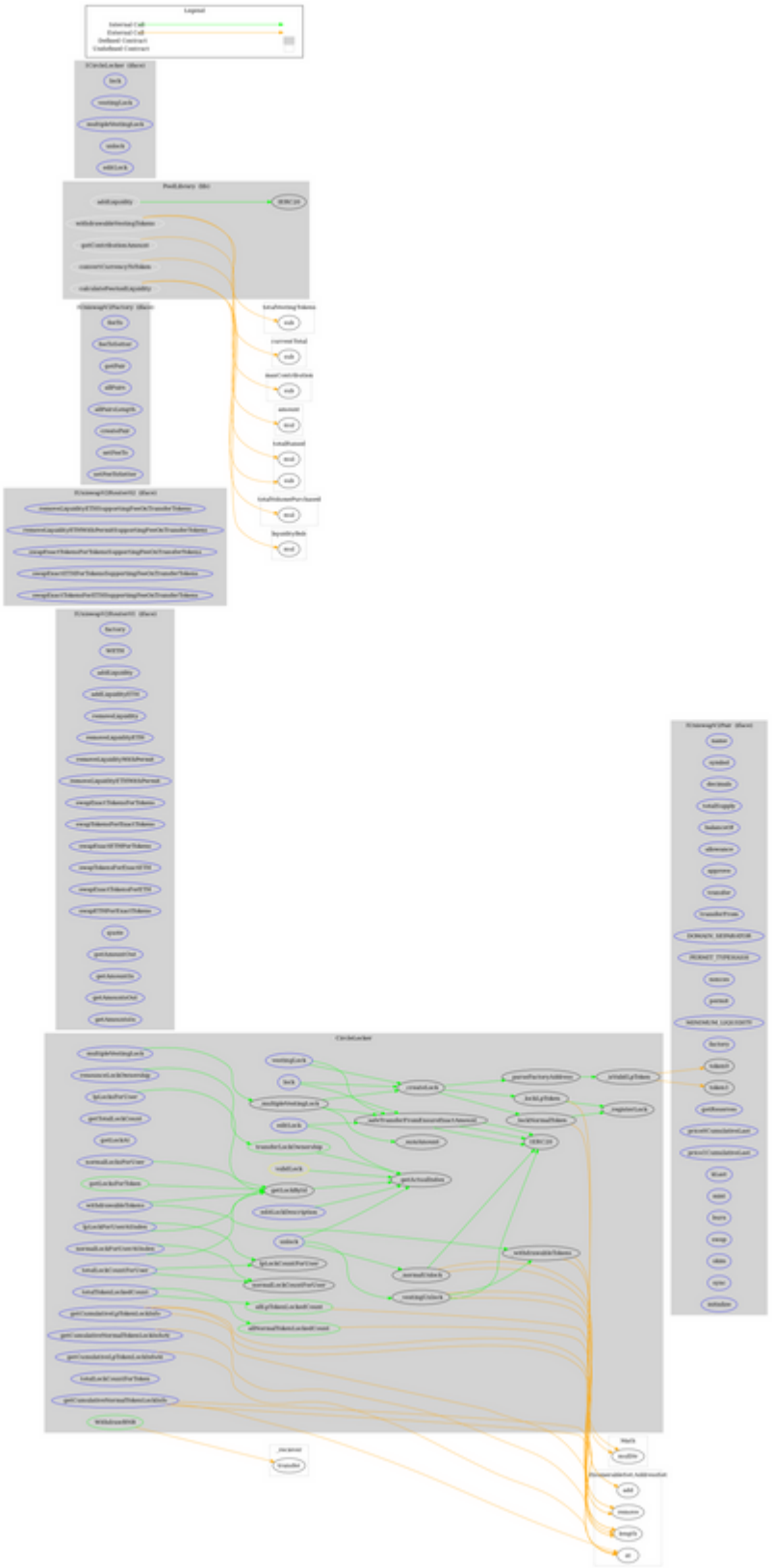
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	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	IUniswapV2 Router01		

	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		-

	getCumulativeNormalTokenLockInfo At	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 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