QuillAudits

Audit Report September, 2021

For

PROTOCOL 3.14

Contents

Overview

PiProtocol
Repository: https://github.com/Pi-Protocol/PI_sol
Branch: Main
Commit: 1e2ccaa0482045c1e2ef1f6927b5462342217323
Fixed In: 74ab94a93388eef0f256eeb2686d88e6791f5a9d

Scope of the Audit

The scope of this audit was to analyze PiProtocol smart contract's codebase for quality, security, and correctness.

pBNB.sol	Wrapped ERC20 with ERC31337 for BNB with dynamic fees.
Pi.sol	Preminted ERC20 contract with transfer gate to apply fees, burn rates, and allow approved pools and routers.
PiEventGate.sol [Will not be utilized in this release]	Preminted ERC20 contract with transfer gate to apply fees, burn rates, and allow approved pools and routers.
PiTransferGate.sol	For any transfer of Pi token, it should pass through these checks and apply rates and checks.
pBNBLiquidity.sol	pBNB<->Pi LP token when deposited mints the Small Circle NFT token to the sender with dynamic fees.
FloorCalculator.sol	To calculate floor calculations checking for locked LP tokens to one in the market.
pBNBDirect.sol	To swap BNB/pBNB token to Pi token and vice versa via PanCakeSwap.

CircleERC1155Token.sol	ERC1155 contract for PI project. Token ID = 1 for Small Circle NFT Token ID = 2 for Big Circle NFT. Small Circle NFT, once bought, is available for a sellout at a 50% rate(dynamic). Big Circle NFT, once bought cannot be sold.
CircleVault.sol	To buy Big Circle from Small Circle NFT, at dynamic rates & also to sell Small Circle NFT to pBNB<->Pi LP tokens.
CircleDirect.sol	To swap BNB/pBNB/Pi token to Small Circle NFT token and vice versa via PanCakeSwap.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return Boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Mythril, Slither, SmartCheck, Surya, Solhint.

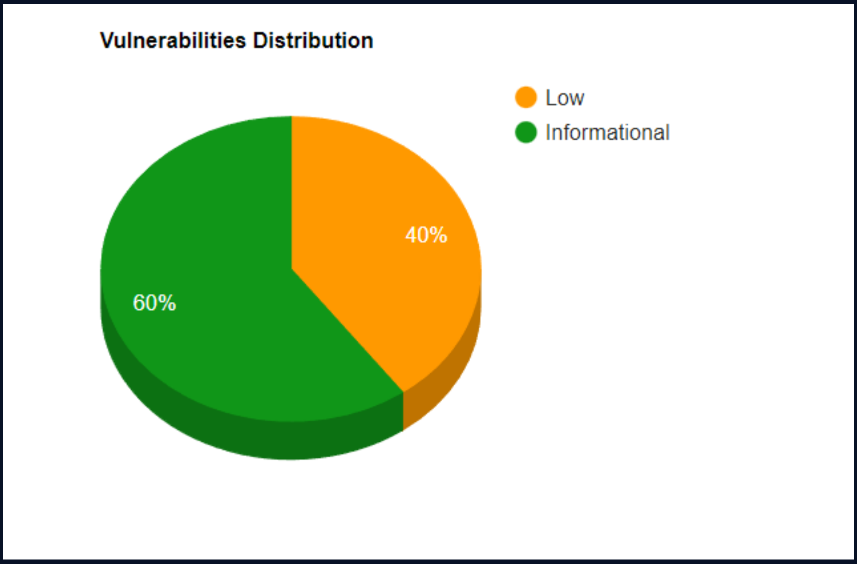
Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.
Low	Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	2	3
Closed	2	3	2	6



Functional Testing Results

The complete functional report is attached below:
Pi Protocol TestCases

Note: UniswapV2Library.**getAmountsOut** and functions depending on it didn't work, while conducting tests on the testnet.

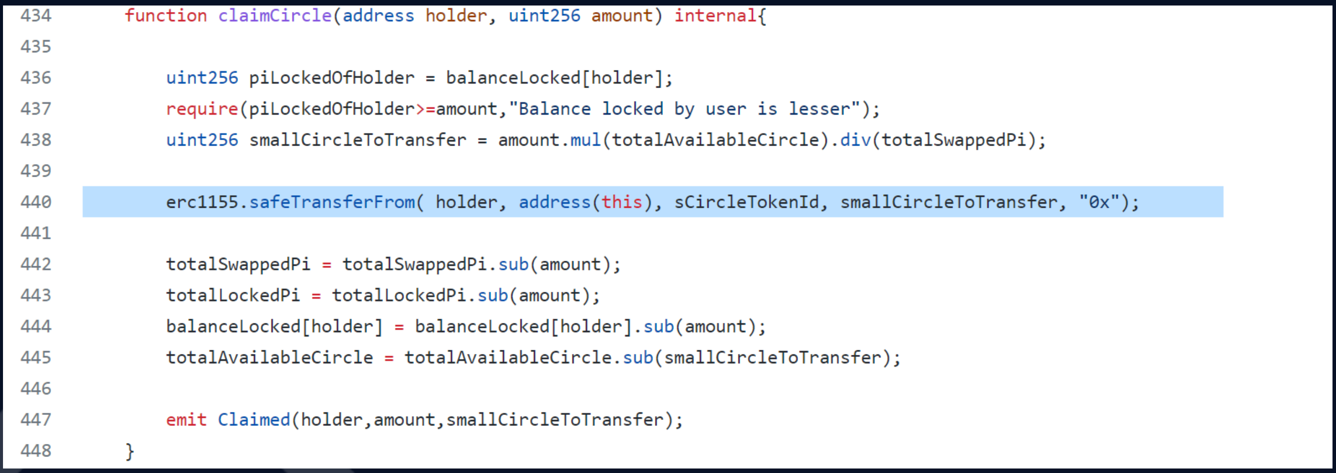
Issues Found

High severity issues

PiEventGate

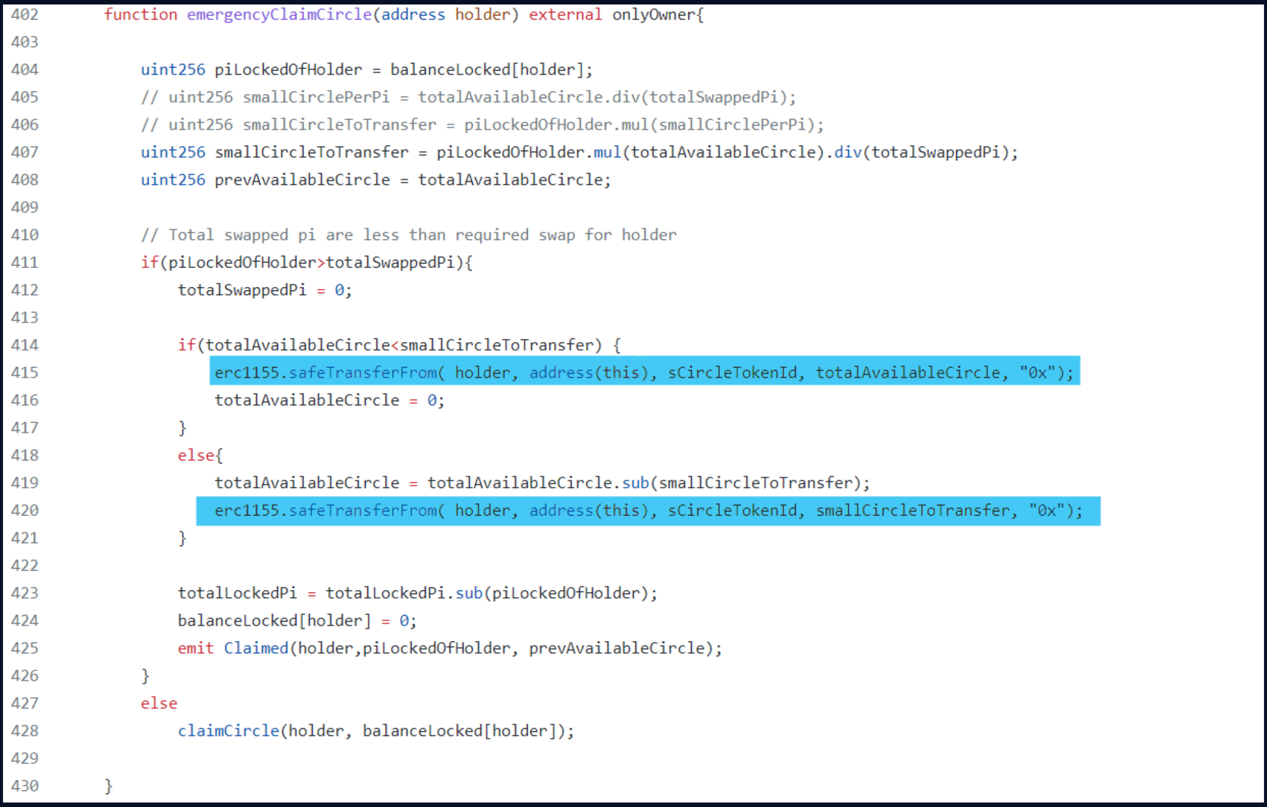
- **[FIXED] Incorrect Business Logic**

  [#L434-448] function **claimCircle**: The intended logic of the function is to distribute small circle NFTs to a holder on the basis of the locked Pi balance of it.
  But the logic implemented here is in reverse order. The function tries to transfer the small circle NFTs from the **holder** to the **PiEventGate contract address**.



```
434     function claimCircle(address holder, uint256 amount) internal{
435 
436         uint256 piLockedOfHolder = balanceLocked[holder];
437         require(piLockedOfHolder>=amount,"Balance locked by user is lesser");
438         uint256 smallCircleToTransfer = amount.mul(totalAvailableCircle).div(totalSwappedPi);
439 
440         erc1155.safeTransferFrom( holder, address(this), sCircleTokenId, smallCircleToTransfer, "0x");
441 
442         totalSwappedPi = totalSwappedPi.sub(amount);
443         totalLockedPi = totalLockedPi.sub(amount);
444         balanceLocked[holder] = balanceLocked[holder].sub(amount);
445         totalAvailableCircle = totalAvailableCircle.sub(smallCircleToTransfer);
446 
447         emit Claimed(holder,amount,smallCircleToTransfer);
448     }
```

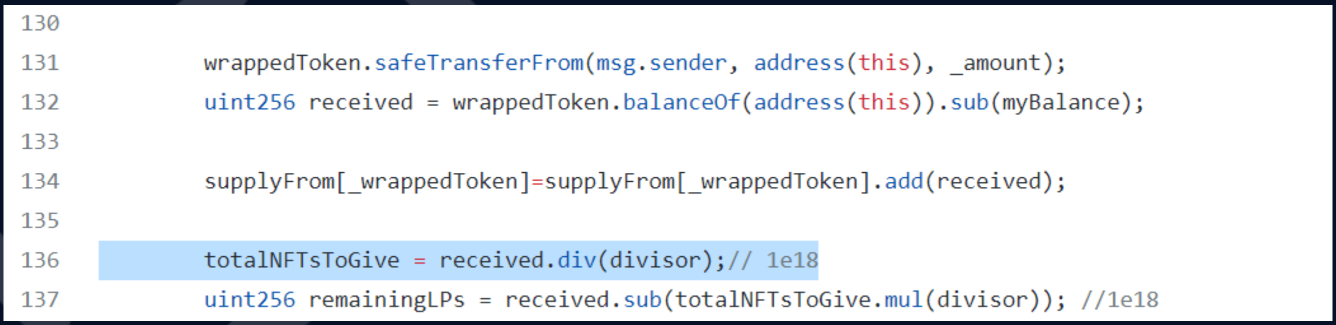
[#L402-430] function **emergencyClaimCircle**, also has the same incorrect logic implemented into it.



```
402     function emergencyClaimCircle(address holder) external onlyOwner{
403
404         uint256 piLockedOfHolder = balanceLocked[holder];
405         // uint256 smallCirclePerPi = totalAvailableCircle.div(totalSwappedPi);
406         // uint256 smallCircleToTransfer = piLockedOfHolder.mul(smallCirclePerPi);
407         uint256 smallCircleToTransfer = piLockedOfHolder.mul(totalAvailableCircle).div(totalSwappedPi);
408         uint256 prevAvailableCircle = totalAvailableCircle;
409
410         // Total swapped pi are less than required swap for holder
411         if(piLockedOfHolder>totalSwappedPi){
412             totalSwappedPi = 0;
413
414             if(totalAvailableCircle<smallCircleToTransfer) {
415                 erc1155.safeTransferFrom( holder, address(this), sCircleTokenId, totalAvailableCircle, "0x");
416                 totalAvailableCircle = 0;
417             }
418             else{
419                 totalAvailableCircle = totalAvailableCircle.sub(smallCircleToTransfer);
420                 erc1155.safeTransferFrom( holder, address(this), sCircleTokenId, smallCircleToTransfer, "0x");
421             }
422
423             totalLockedPi = totalLockedPi.sub(piLockedOfHolder);
424             balanceLocked[holder] = 0;
425             emit Claimed(holder,piLockedOfHolder, prevAvailableCircle);
426         }
427         else
428             claimCircle(holder, balanceLocked[holder]);
429
430     }
```

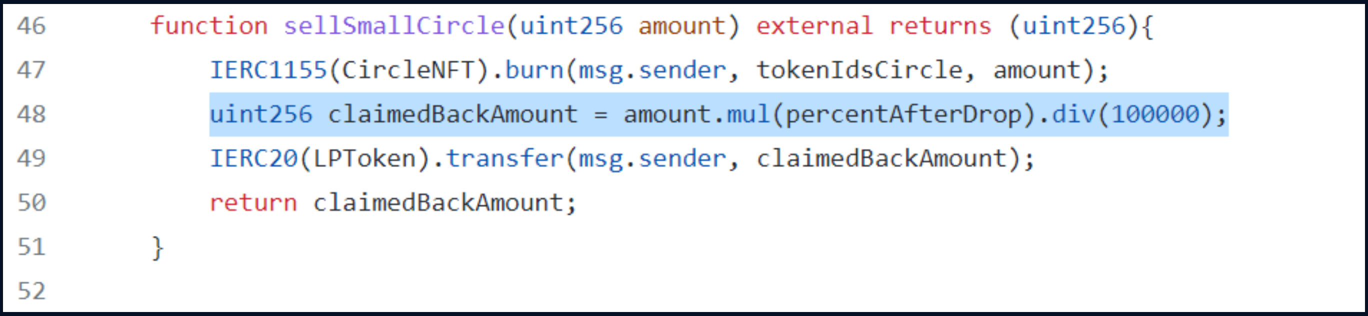
- **[FIXED] No decimal consideration leads to Incorrect token transfers**

  [#L124-156] function **depositTokens** in contract **pBNBLiquidity** calculates **totalNFTsToGive** by dividing **LP tokens** deposited with a set **divisor**



```
130
131         wrappedToken.safeTransferFrom(msg.sender, address(this), _amount);
132         uint256 received = wrappedToken.balanceOf(address(this)).sub(myBalance);
133
134         supplyFrom[_wrappedToken]=supplyFrom[_wrappedToken].add(received);
135
136         totalNFTsToGive = received.div(divisor);// 1e18
137         uint256 remainingLPs = received.sub(totalNFTsToGive.mul(divisor)); //1e18
```

But while refunding LP tokens, circleVault doesn't consider token decimals while transferring



```
    function sellSmallCircle(uint256 amount) external returns (uint256){
        IERC1155(CircleNFT).burn(msg.sender, tokenIdsCircle, amount);
        uint256 claimedBackAmount = amount.mul(percentAfterDrop).div(100000);
        IERC20(LPToken).transfer(msg.sender, claimedBackAmount);
        return claimedBackAmount;
    }
```

As a result, incorrect token transfers may happen

Consider a scenario as:

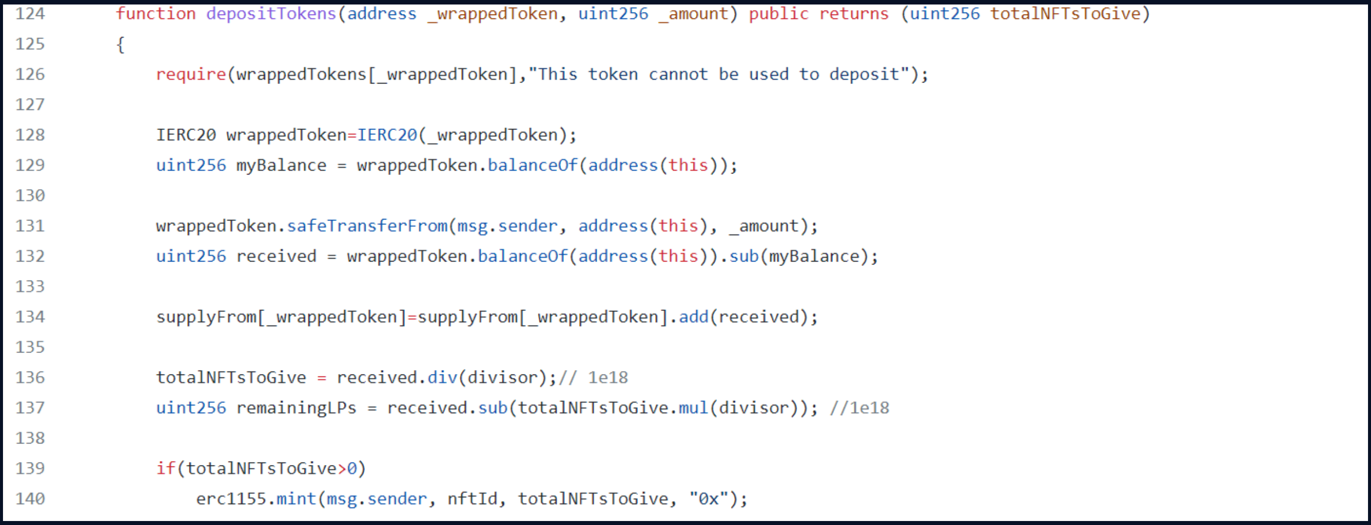
User deposits LP tokens having 18 decimals, the **totalNFTsToGive** with the current **divisor** as **1e14**, will be calculated in the multiples of **1e4**. Now if a user tries to sell these small circle NFTs, the refund **claimedBackAmount** will also be calculated in the multiples of **1e4** and not with the correct token decimals.

Medium severity issues

pBNBLiquidity & CircleVault

- **[FIXED] LP Token Mismatch while buying/selling NFTs**

  [#L124-156] function **depositTokens** in contract **pBNBLiquidity** mints small circle NFTs to a user, when it deposits a certain wrappedToken/ LP Token.



```
124     function depositTokens(address _wrappedToken, uint256 _amount) public returns (uint256 totalNFTsToGive)
125     {
126         require(wrappedTokens[_wrappedToken],"This token cannot be used to deposit");
127 
128         IERC20 wrappedToken=IERC20(_wrappedToken);
129         uint256 myBalance = wrappedToken.balanceOf(address(this));
130 
131         wrappedToken.safeTransferFrom(msg.sender, address(this), _amount);
132         uint256 received = wrappedToken.balanceOf(address(this)).sub(myBalance);
133 
134         supplyFrom[_wrappedToken]=supplyFrom[_wrappedToken].add(received);
135 
136         totalNFTsToGive = received.div(divisor);// 1e18
137         uint256 remainingLPs = received.sub(totalNFTsToGive.mul(divisor)); //1e18
138 
139         if(totalNFTsToGive>0)
140             erc1155.mint(msg.sender, nftId, totalNFTsToGive, "0x");
```

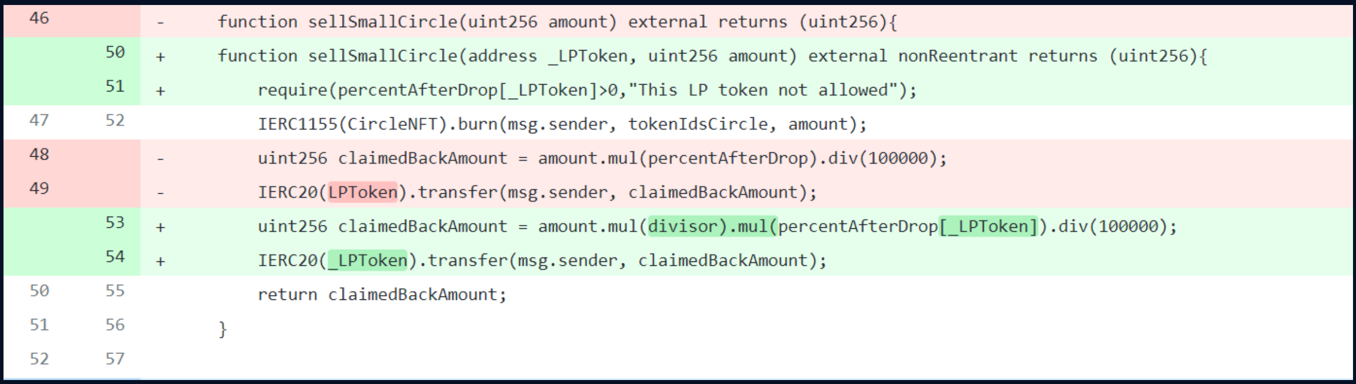
The contract expects to have allowed more than one LP Token to be deposited.

But the **CircleVault** has the LP Token fixed into it. As a result, there may be a LP Token mismatch.

Possible Cases:

A user deposits LP tokens **X** and **Y** to get small circle NFTs, but it can get a refund in only one of the LP tokens set into the circle vault and the other token will not be refundable.
Also, considering the above case, a user may get a refund of more than the maximum refundable amount for a LP Token for that user, as the user may have an NFT balance purchased from two different LP token balances.

Status Update: The fixes have been applied in commit 40a...f47 as



```
-   function sellSmallCircle(uint256 amount) external returns (uint256){
+   function sellSmallCircle(address _LPToken, uint256 amount) external nonReentrant returns (uint256){
+       require(percentAfterDrop[_LPToken]>0,"This LP token not allowed");
        IERC1155(CircleNFT).burn(msg.sender, tokenIdsCircle, amount);
-       uint256 claimedBackAmount = amount.mul(percentAfterDrop).div(100000);
-       IERC20(LPToken).transfer(msg.sender, claimedBackAmount);
+       uint256 claimedBackAmount = amount.mul(divisor).mul(percentAfterDrop[_LPToken]).div(100000);
+       IERC20(_LPToken).transfer(msg.sender, claimedBackAmount);
        return claimedBackAmount;
    }
```

With this, now a user can select the LP Token it wants the refund in. It fixes the the first case, but the second case still persists.

Possible Cases:

1. The user can deposit LP Token X and Y to mint small circle NFTs, but while selling small circle NFTs, a user may opt for a refund in one of the LP tokens and may get a refund of more than the maximum refundable amount for that LP Token for the user.

2. A user may opt for a refund in an LP Token which it has never deposited to mint NFTs, which may create a token imbalance.

Status Update: The contracts are now restricted to support only one LP Token

CircleERC1155Token

- **[FIXED] Create may reset tokenSupply of an existing id**

  [#L38-56] function **create** doesn't check for existing Token IDs. As a consequence, it may reset a **tokenSupply** to the **initialSupply** supplied.

  Practical Scenario:

  Consider, users buying **big circle NFTs** from the vault by depositing **small circle NFTs**.
  Now if **create** function is called either accidentally or on purpose, it will modify the **tokenSupply** of **big circle NFTs**.

  Recommendation: Consider adding a check for existing tokenIDs.

CircleDirect

- **[FIXED] Considering the fixes applied in commit 40a...f47**

  The signature to call the **sellSmallCircle** function of the circle vault now needs to be updated as the function now expects LP Token as an argument.

Low level severity issues

- **[FIXED] Not Using SafeMath for arithmetic operations**

  Several Occurrences have been found, where the code uses +,/,-,* for arithmetic operations instead of the **SafeMath** library. The contracts use solidity compiler version 0.7.X which doesn't protect against overflows and underflows.

  Some of these occurrences are:

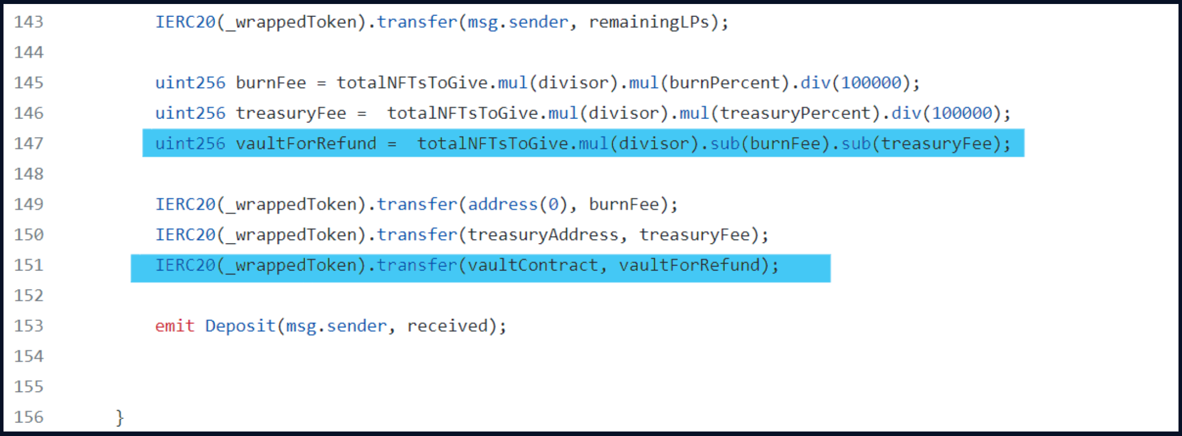
Line Numbers	Contracts
110, 116	**FloorCalculator**
185, 213, 252	**PiEventGate**
213, 218, 222	**PiTransferGate**

  Recommendation: Highly recommended to use SafeMath for **better code readability**, as it has been observed that the library has already been imported and used at most of the places and also to avoid any **edge-case scenarios for underflows/overflows**.

pBNBLiquidity & CircleVault

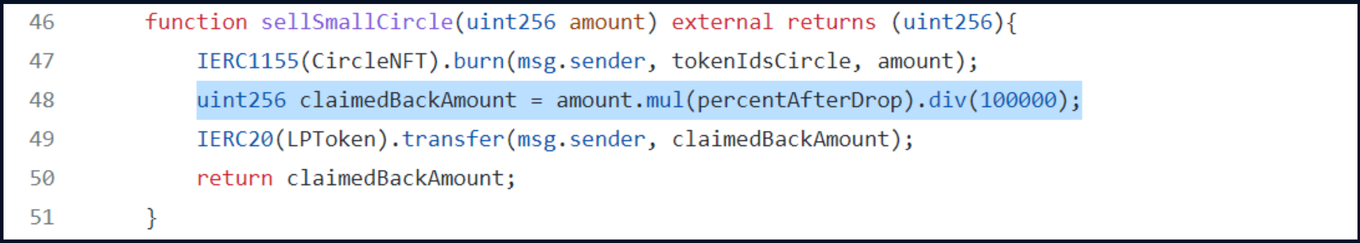
- **[Acknowledged] Incorrect percentAfterDrop may lock LP tokens into Vault or may transfer more than the maximum refundable amount for a user**

  [#L124-156] function **depositTokens** in contract **pBNBLiquidity** allows a user to deposit **LP tokens** and get **small circle NFTs**. Function transfers, deposited LP tokens to **circle vault** after subtracting a **burn** and **treasury** percentage from it,



```
        IERC20(_wrappedToken).transfer(msg.sender, remainingLPs);

        uint256 burnFee = totalNFTsToGive.mul(divisor).mul(burnPercent).div(100000);
        uint256 treasuryFee =  totalNFTsToGive.mul(divisor).mul(treasuryPercent).div(100000);
        uint256 vaultForRefund =  totalNFTsToGive.mul(divisor).sub(burnFee).sub(treasuryFee);

        IERC20(_wrappedToken).transfer(address(0), burnFee);
        IERC20(_wrappedToken).transfer(treasuryAddress, treasuryFee);
        IERC20(_wrappedToken).transfer(vaultContract, vaultForRefund);

        emit Deposit(msg.sender, received);


    }
```

so as to allow a user to sell small circle NFTs and get a considerable amount of LP tokens back as a refund.



```
    function sellSmallCircle(uint256 amount) external returns (uint256){
        IERC1155(CircleNFT).burn(msg.sender, tokenIdsCircle, amount);
        uint256 claimedBackAmount = amount.mul(percentAfterDrop).div(100000);
        IERC20(LPToken).transfer(msg.sender, claimedBackAmount);
        return claimedBackAmount;
    }
```

Incorrect **percentAfterDrop** may refund a user an amount more than the maximum refundable amount or even lock some LP tokens of a user.

Consider, (burn + treasury) percent as 60%, so the vault will be getting 40% of LP tokens to refund the user.

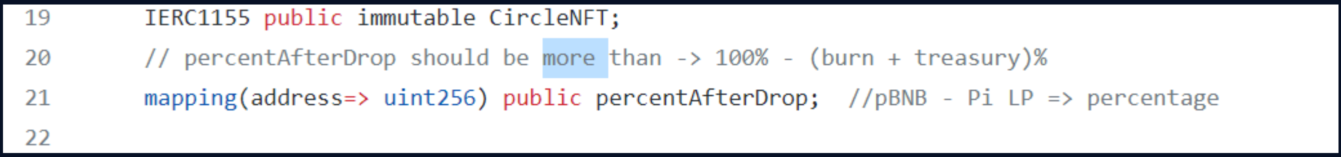
If the **percentAfterDrop** is more than 40%, it will transfer tokens to the user more than what it should.

Also, if **percentAfterDrop** is less than 40%, then even after selling all the small circle NFTs, a user will be getting fewer LP tokens than intended.

Recommendation: A check can be introduced so as to make sure the **percentAfterDrop** stays equal to **(100% - (burn + treasury)%)**

Dev Comments/Status Update: It has been acknowledged that the values will be set, keeping the recommended checks in mind.

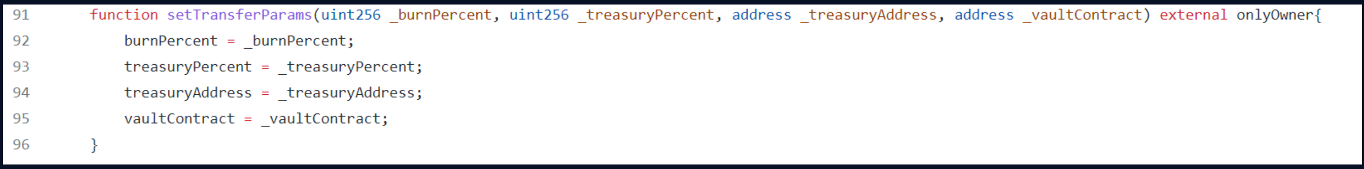
We recommend fixing the comment at #L20 to replace **more** with **less** to avoid any future conflicts and confusion.



```
    IERC1155 public immutable CircleNFT;
    // percentAfterDrop should be more than -> 100% - (burn + treasury)%
    mapping(address=> uint256) public percentAfterDrop;  //pBNB - Pi LP => percentage

```

pBNBLiquidity

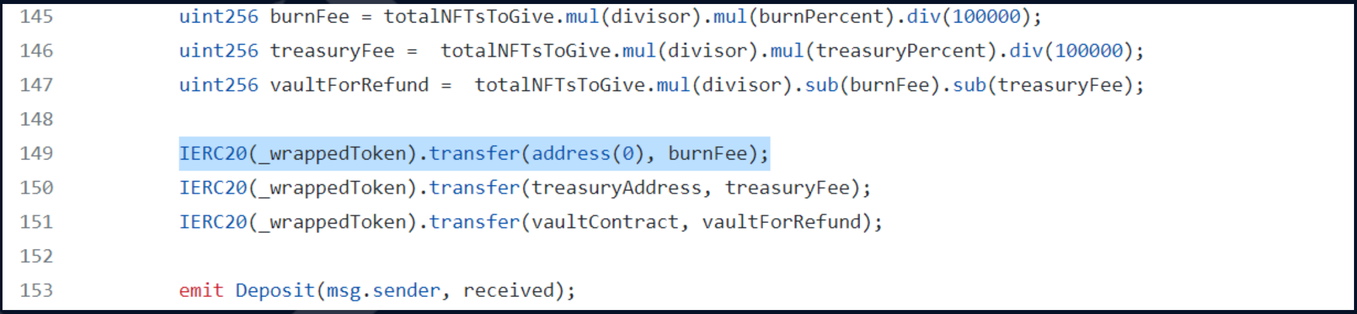
- [FIXED] A require check can be added so as to make sure **burnPercent** and **treasuryPercent** should be less than or equal to **100%**



```
    function setTransferParams(uint256 _burnPercent, uint256 _treasuryPercent, address _treasuryAddress, address _vaultContract) external onlyOwner{
        burnPercent = _burnPercent;
        treasuryPercent = _treasuryPercent;
        treasuryAddress = _treasuryAddress;
        vaultContract = _vaultContract;
    }
```

Status Update: A require check has been added to individually limit **burnPercent** and **treasuryPercent** to not exceed 100%, but they can still add up to be more than 100%. It is recommended to add an extra check, so as to limit (burn + treasury) to not exceed 100%.

- [Acknowledged] Transferring tokens to Zero Address will not reduce the totalSupply



```
        uint256 burnFee = totalNFTsToGive.mul(divisor).mul(burnPercent).div(100000);
        uint256 treasuryFee =  totalNFTsToGive.mul(divisor).mul(treasuryPercent).div(100000);
        uint256 vaultForRefund =  totalNFTsToGive.mul(divisor).sub(burnFee).sub(treasuryFee);

        IERC20(_wrappedToken).transfer(address(0), burnFee);
        IERC20(_wrappedToken).transfer(treasuryAddress, treasuryFee);
        IERC20(_wrappedToken).transfer(vaultContract, vaultForRefund);

        emit Deposit(msg.sender, received);
```

Dev Comments/ Status Update: This is desired and there is no advantage to burn LPs

Informational

- [Acknowledged] **Public functions that are never called by the contract should be declared external to save gas.**
- [Acknowledged] **ERC20 approve() race:**

  The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.

  Same issues have been found in the function safeApprove() and has been deprecated.

  Reference:

  https://eips.ethereum.org/EIPS/eip-20

PiEventGate

- [Acknowledged] [#L110] MIN_PI is currently 0. Make sure to set an appropriate value prior to deployment of the contract.
- [FIXED] [#L261] **isHolderAddress[recipient]==false** in [#L236-272]function **handleZap**

  [#L389] **AdminByAddress[msg.sender] == true** in [#L386-396] function **claimCircleForHolder**

  [#L468] **AdminByAddress[msg.sender] == true** in [#L467-470] modifier **onlyAdmin** compares with a boolean constant, whereas **Boolean constants can be used directly and do not need to be compared to true or false.**

pBNBDirect

- [FIXED] **[#L152] pBNB.isIgnored(msg.sender)==false**

  in [#L130-169]function **sell** compares with a boolean constant, whereas **Boolean constants can be used directly and do not need to be compared to true or false.**

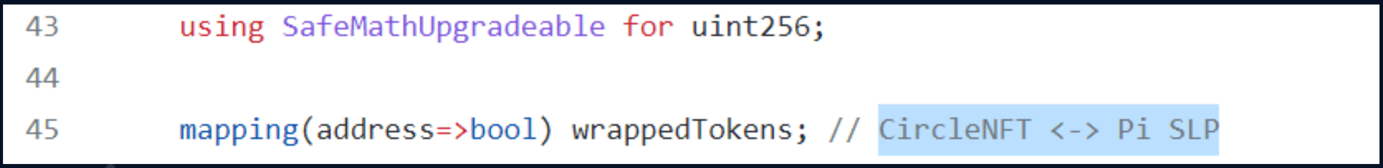
CircleDirect

- [FIXED] [#L309] **pBNB.isIgnored(msg.sender)==false**

  in [#L292-327]function **easySellSmallCircleToBNB** compares with a boolean constant, whereas **Boolean constants can be used directly and do not need to be compared to true or false**

pBNBLiquidity

- [FIXED] **Wrong Comment: wrappedToken** is an LP pair of pBNB < - > Pi,



```
    using SafeMathUpgradeable for uint256;

    mapping(address=>bool) wrappedTokens; // CircleNFT <-> Pi SLP
```

CircleVault

- [FIXED] The contract inherits **ReentrancyGuard**, but no occurrences have been found for **nonReentrant** modifier.

CircleERC1155Token

- [FIXED] **NFTs can be minted without a URI**

  NFTs can be minted without **create** function ever called, as a result, there will be no URI to track.

Closing Summary

Several issues of high, medium and low severity have been reported during the audit. Some suggestions are also reported to improve the code quality and save gas fees. All of the high and medium issues have been fixed.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or endorsement of the **PI platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the PI Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
September, 2021

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com