QuillAudits

Audit Report
June, 2022

For

PATHFUND

Table of Content

Executive Summary

Project Name	PathFund
Overview	The smart contracts are intended to be used for launchpad where various launchpads can be launched and configured through LaunchFactory contract. The collected fund can be locked in KKLocker which can only be withdrawn by the contract owner. PATHlaunchpad is used for the core logic for token deposit, claim, vesting, finalizing launchpad and other configs.
Timeline	9th May, 2022 to 21st May, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse PathFund codebase for quality, security, and correctness.
Source Code	Source Code Zip File was Provided by PathFund Team
Fixed In	Updated Source Code Zip File was Provided by PathFund Team

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	3
Partially Resolved Issues	0	0	1	0
Resolved Issues	1	1	4	12

Types of Severities

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 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 issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- 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

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - KKlocker

High Severity Issues

No Issues Found

Medium Severity Issues

No Issues Found

Low Severity Issues

A.1 Missing check for timestamp

Line	Function - updateLock and constructor
14	`constructor( uint256 _locked_until, address own, address lpA ) { locked_until = _locked_until; ownerz = own; lpToken = lpA; }`
26	`// update timer once it's over` ftrace | funcSig `function updateLock(uint256 _newTime) external onlyOwnerz { require(locked_until <= _newTime, "new date cannot be in the past"); locked_until = _newTime; }`

Description

While updating the value of locked_until, there is no check for the timestamp if it is a past timestamp. This can be updated to a past value to keep the locker open always.

Remediation

To solve the issue, a check should be placed in the function to ensure the timestamp has not yet arrived.

Status

Fixed

Informational Issues

A.2 Unlocked pragma (pragma solidity ^0.8.13)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might negatively introduce bugs that affect the contract system.

Recommendation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or unaudited third-party libraries. If necessary, refactor the current code base only to use stable features.

Status

Fixed

A.3 Misleading error message

Line	Function - updateLock

34	`// update timer once it's over`
	`ftrace | funcSig`
	`function updateLock(uint256 _newTime) external onlyOwnerz {`
	`  require(locked_until <= _newTime, "new date cannot be in the past");`
	`  locked_until = _newTime;`
	`}`

Description

In the updateLock function the check is to see whether the last locked_until value is less than new time or not but the error says “new date cannot be in the past”.

Remediation

It is recommended to update the error message.

Status

Fixed

A.4 General Recommendation

The contract uses BEPz20 which can be replaced with a standard IBEP20 interface and reused in PATHlaunchpad, as well as the purpose of using it, is for interacting with existing token contracts and not inheritance.

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Status

Acknowledged

B. Contract - PATHlaunchpad

High Severity Issues

No Issues Found

Medium Severity Issues

B.1 Check-Effect-Interaction pattern not followed

Line	Function - claim

57	`if (claimSentCounter + 75 < tPart) {`
	`  for (uint256 i = claimSentCounter; i < claimSentCounter + 75; i++) {`
	`  address temp_user = EnumerableSet.at(partecipant, i);`
	`  uint256 temp_amnt = userDB[temp_user].claimableToken;`
	`  userDB[temp_user].claimableToken = 0;`
	`  require(BEP20(token).transfer(temp_user, temp_amnt));`
	`  claimSentCounter += 1;`
	`  }`
	`} else {`
	`  for (uint256 i = claimSentCounter; i < tPart; i++) {`
	`  address temp_user = EnumerableSet.at(partecipant, i);`
	`  uint256 temp_amnt = userDB[temp_user].claimableToken;`
	`  userDB[temp_user].claimableToken = 0;`
	`  require(BEP20(token).transfer(temp_user, temp_amnt));`
	`  claimSentCounter += 1;`
	`  }`

65	

Description

Calling BEP20(token).transfer() is a call to an external contract. If there are vulnerable external calls, reentrancy attacks could be conducted because these two functions have state updates and events emitted after external calls. The audit scope would treat the third-party implementation as a black box and assume its functional correctness. However, third parties may be compromised in the real world, leading to lost or stolen assets.

Remediation

As per the solidity security recommendation, the functions should first update the contract states and then interact with external contracts.
Please refer to solidity documentation here: *https://docs.soliditylang.org/en/develop/security-considerations.html#use-the-checks-effects-interactions-pattern*

Status

Fixed

Low Severity Issues

B.2 Add external modifier instead of public

Description

It is recommended to use external access modifier instead of public for functions which are not called from the contract.

Recommendation

It is recommended to update nextPhase() and set_tierLimit() function to external from public

Status

Partially Resolved

B.3 Usage Of transfer Instead Of safeTransfer

Line	Function - emergency_killPreSale
354	`}else{ for(uint i=0; i<hoymanyToProcess⬆; i++){ address tmpvalue = EnumerableSet.at(partecipant,i); EnumerableSet.remove(partecipant,tmpvalue); BEP20(pair).transfer(tmpvalue, userDB[tmpvalue].amountDeposit); } }`

Description

The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks. In effect, the transaction would always succeed, even if the token transfer didn't.

Remediation

Use the safeTransfer function from the safeERC20 Implementation or put the transfer call inside an assert or require to verify that it returned true.

Status

Fixed

Informational Issues

B.4 State Variable Default Visibility

Description

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Recommendation

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables. Ref: *https://swcregistry.io/docs/SWC-108*

Status

Fixed

B.5 General Recommendation

The contract contains a lot of redundant code which can be reused to reduce the transaction and deployment gas costs. (Check deposit(), claim(), and vestiti() functions)

Checks should be applied on inputs of all the functions which are accessed by the creator to avoid setting exceptional values and have transparency.

The contract uses BEP20 abstract contract which can be replaced with a standard IBEP20 interface.

The contracts do not follow naming conventions and an official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Status

Acknowledged

C. Contract - PathLaunchpad_factory

High Severity Issues

C.1 Usage Of transfer Instead Of safeTransfer

Line	Function - transferForeignToken

153	ftrace | funcSig
	function transferForeignToken(address _token↑, uint _value↑) external onlyOwner returns(boo
	if(_value↑ == 0) {
	_value↑ = BEP20(_token↑).balanceOf(address(this));
	} else {
	_sent↑ = BEP20(_token↑).transfer(owner(), _value↑);
	}
	emit stuckedTokenRecovered(_token↑, _value↑);
	}

Description

The ERC20 standard token implementation functions return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks. In effect, the transaction would always succeed, even if the token transfer didn't.

Remediation

Use the safeTransfer function from the safeERC20 Implementation or put the transfer call inside an assert or require to verify that it returned true.

Status

Fixed

Medium Severity Issues

No Issues Found

Low Severity Issues

C.2 Incorrect if-else condition

Line	Function - transferForeignToken

153	ftrace | funcSig
	function transferForeignToken(address _token⬆, uint _value⬆) external onlyOwner returns(boo
	if(_value⬆ == 0) {
	_value⬆ = BEP20(_token⬆).balanceOf(address(this));
	} else {
	_sent⬆ = BEP20(_token⬆).transfer(owner(), _value⬆);
	}
	emit stuckedTokenRecovered(_token⬆, _value⬆);
	}

Description

In the function, it checks if the _value is 0, it updates the _value variable to be equal to the balance of the contract and later emits the event with this value. Ideally, it should be transferring the whole balance to the owner, if the balance is non-zero. Otherwise, it will only work in case value is provided and the if() code will be irrelevant.

Remediation

It is recommended to consider the below implementation to avoid such a situation.

```
ction transferForeignToken(address _token⬆, uint256 _value⬆)
 external
 onlyOwner
 returns (bool _sent⬆)

 _sent⬆ = _value⬆ == 0
     ? BEP20(_token⬆).transfer(
         owner(),
         BEP20(_token⬆).balanceOf(address(this))
       )
     : BEP20(_token⬆).transfer(owner(), _value⬆);
 emit stuckedTokenRecovered(_token⬆, _value⬆);
```

Status

Fixed

C.3 Add external modifier instead of public

Public functions that are never called by the contract should be declared external to save gas. Make createPsale() function external.

Status

Fixed

Informational Issues

C.4 State Variable Default Visibility

Description

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Recommendation

Variables can be specified as being public, internal, or private. Explicitly define visibility for all state variables. Ref: *https://swcregistry.io/docs/SWC-108*

Status

Fixed

C.5 General Recommendation

Line	Function - createPsale
62	`uint tokWfee = tokNoFee + ((tokNoFee*feeTk)/100);` `if(tokenFeeActive){` `require(BEP20(addressInfo⬆[0]).transferFrom(msg.sender, address(pool), tokWfee));` `}else{` `require(BEP20(addressInfo⬆[0]).transferFrom(msg.sender, address(pool), tokNoFee));` `}`

Checks should be applied to inputs of all the functions which are accessed by the owner to avoid setting exceptional values and have transparency. The contract uses BEP20 abstract contract which can be replaced with a standard IBEP20 interface.
Line number 62 can be included inside if condition to avoid extra storage costs every time as below.

```
uint256 tokNoFee = _tokAm + ((_tokAm * uintInfo⬆[5]) / 100);
if (tokenFeeActive) {
    uint256 tokWfee = tokNoFee + ((tokNoFee * feeTk) / 100);
    require(
        BEP20(addressInfo⬆[0]).transferFrom(
            msg.sender,
            address(pool),
            tokWfee
        )
    );
} else {
    require(
        BEP20(addressInfo⬆[0]).transferFrom(
            msg.sender,
            address(pool),
            tokNoFee
        )
    );
}
```

It is recommended to remove the console.sol contract and console.log statements to avoid additional deployment costs.
The contracts do not follow naming conventions and an official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Status

Acknowledged

D. Contract - spawnerL

High Severity Issues

No Issues Found

Medium Severity Issues

No Issues Found

Low Severity Issues

No Issues Found

Informational Issues

D.1 Unlocked pragma (pragma solidity ^0.8.13)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might negatively introduce bugs that affect the contract system.

Recommendation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or unaudited third-party libraries. If necessary, refactor the current code base only to use stable features.

Status

Fixed

D.2 No error messages

Line	Function

16
 	
22
	ffrace | funcSig
	`function spawnaLockr(uint liqlock↑, address creatorz↑, address pairz↑) external returns(address){`
	`require(msg.sender == factory);`
	`KKLocker lockr = new KKLocker(liqlock↑, creatorz↑, pairz↑);`
	`emit lockerSpawned(address(lockr), creatorz↑, pairz↑);`
	`return address(lockr);`
	`}`
	ffrace | funcSig
	`function set_factory( address adr↑) public {`
	`require(msg.sender == owner);`
	`factory = adr↑;`
	`}`

Description

In the spawnaLockr function and set_factory function, the check does not contain an error message which is needed to identify the error in production.

Remediation

It is recommended to add the error message.

Status

Fixed

D.3 Function input parameters lack check

Description

In the set_factory function, there is no check to verify if the factory address is not invalid.

Recommendation

It is recommended to add a check to verify if adr != address(0).

Status

Fixed

D.4 General Recommendation

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts. For Example, the Contract name should start with a capital letter.

Status

Acknowledged

E. Contract - spawnerP

High Severity Issues

No Issues Found

Medium Severity Issues

No Issues Found

Low Severity Issues

No Issues Found

Informational Issues

E.1 Unlocked pragma (pragma solidity ^0.8.13)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might negatively introduce bugs that affect the contract system.

Recommendation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or unaudited third-party libraries. If necessary, refactor the current code base only to use stable features.

Status

Fixed

E.2 No error message

Description

In all the functions, the check does not contain an error message which is needed to identify the error in production.

Recommendation

It is recommended to add the error message.

Status

Fixed

E.3 Function input parameters lack check

Line	Function - set_factory and set_pathAddress

34	ftrace | funcSig
	`function set_factory( address adr⬆) public {`
	`  require(msg.sender == owner);`
	`  factory = adr⬆;`
	`}`

38	ftrace | funcSig
	`function set_pathAddress( address adr⬆) public {`
	`  require(msg.sender == owner);`
	`  pathAddress = adr⬆;`
	`}`

Description

In the functions, there is no check to verify if the address is not invalid.

Remediation

It is recommended to add a check to verify if adr != address(0).

Status

Fixed

E.4 General Recommendation

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts. For Example, the Contract name should start with a capital letter.

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below:

- ✓ Should be able to deploy and create launchpad.
- ✓ Should be able to deposit and withdraw
- ✓ Should be able to finalize and vest
- ✓ Should be able to claim tokens
- ✓ Should revert if configurational functions are not called by owner/creator
- ✓ Should be able to withdraw from KKLocker contract only if unlock time is reached.

```
 Launchpad testing...
PRESALE 1 TEST START
pathC deployed:  0xf8fFbEc3aBaeBE53880C3C0F3701A8a8b767556b
  ✓ deploy path and create pair
launchapd deployed 0xE3FB3F714852B66Bf09fc546D4A0A4E438dDEAD0
  ✓ deploy launchpad and spawners
token deployed:  0xE29Bdb43ee785A96CB3763D37Ac69EAfc2CF123A
  ✓ deploy  Token presale1
presale1 address o/ 0x174B94Af0fFB8D1b2fD4e00756F62190E30b596a
0
0x174B94Af0fFB8D1b2fD4e00756F62190E30b596a
  ✓ create presale 1
  ✓  phase 1 deposit and withdraw
phase 2
50 user deposited 1.5 bnb and 1 4.5.. bnb raised: 79.5
  ✓ 50 user deposit in the presale
raised after user deposit: 80.0
raised after user withdraw: 79.5
  ✓  user deposit and withdraw
0 [before]balance token random user
  ✓  finalize
150000000000000 [after claim]balance token random user
PRESALE 1 TEST DONE
  ✓  claim
PRESALE 2 TEST START
token deployed:  0x036F4C919487853a16aEc8694d1835D765791d4c
  ✓ deploy  Token presale2
presale 2 address o/ 0x21c78cA8F3E6650DA1859B1EAa327a5ACF57f6FB
```

```
1
0x21c78cA8F3E6650DA1859B1EAa327a5ACF57f6FB
[
 BigNumber { value: "750000000000000000000" },
 BigNumber { value: "1500000000000000000000" },
 BigNumber { value: "0" },
 BigNumber { value: "1650671877" },
 BigNumber { value: "1654206969" },
 BigNumber { value: "1656836712" },
 BigNumber { value: "51" }
] aaaaaaaaaaaaa
  ✓ create presale
balance tk whale 0 -- 5000000000000000000,5
  ✓  phase 1 deposit and withdraw
phase 2
30 user deposited 2.9k usd.. usd raised: 87000.0
  ✓ 30 user deposit in the presale
0 [before]balance token random user
  ✓  finalize
  1st claim
  4060000000000000 [after claim]balance token random user
  1 MONTH LATER...
  2nd claim
  PRESALE 2 TEST DONE
    ✓  claim vesting |
```

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the PathFund. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found. In the End, Pathfund team resolved all Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the PathFund 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 PathFund Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2022

For

PATHFUND

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com