QuillAudits

Audit Report
August, 2021

BLOCKPAD

Contents

Scope of Audit

The scope of this audit was to analyze and document the Blockpad Token smart contract codebase for quality, security, and correctness.

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
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

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 analyzed 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

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.

High severity issues

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 level severity issues

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 level severity issues

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	1	2
Closed	0	0	3	5

Introduction

During the period of **August 3, 2021 to August 18, 2021** - QuillAudits Team performed a security audit for Blockpad smart contracts.

The code for the audit was taken from the following official link: https://github.com/theblockpad/BPADVault

Note	Date	Commit hash
Version 1	August	-
Version 2	August	be54d3810d7d0a5fc06d9426b4d5c9706c36610e
Version 3	August	6bfe159a79b6c756f86fe18571cb1dadcc4ba3eb

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found

Medium severity issues

No issues were found

Low level severity issues

1. User can get vesting discount for single escrow

Line	Code
142 - 145	uint256 fee = _amounts.length.mul(bnbFee).sub(_amounts.length.mul(bnbFee).mul(vestingDiscount) .div(100));

Description

The lockTokens method creates a single escrow with user funds without any vesting discount. In contrast, the createMultipleLocks method rewards the user with a discount in fee for creating multiple escrows.

Since the createMultipleLocks does not check for a minimum number of escrows before applying a discount, this allows any user to create one escrow using this method and still get a discount. In other words, there is no incentive for the user to use lockTokens method for creating escrows.

Remediation

Consider checking for a minimum number of escrows before applying a vesting discount in createMultipleLoks methods.

Status: Fixed

This has been fixed in commit be54d3810d7d0a5fc06d9426b4d5c9706c36610e.

2. Expects the transfer methods to always revert/return

Line	Code
418	require(IERC20(_tokenAddress).transferFrom(msg.sender, address(this), _amount));
128	require(IERC20(tokenAddress).transfer(msg.sender, amount));
178-184	require(IERC20(_tokenAddress).transferFrom(msg.sender, address(this), _amount));

Description

The functions lockTokens(), withdrawTokens() and createMultipleLocks() all expect the transfer and transferFrom methods to always throw or return a boolean value on success/failure. There are some ERC20 token implementations that do neither of these, resulting in the require check to always fail.

Remediation

Consider adding additional validations to ensure the tokens without a return value are also supported. OpenZeppelin's safeTransfer and safeTransferFrom provide similar functionality, and it can be used to avoid custom implementations.

Status: Fixed

This has been fixed in commit 6bfe159a79b6c756f86fe18571cb1dadcc4ba3eb.

3. Potential DoS because of block gas limit

Line	Code

112	uint256 arrLength = depositsByWithdrawalAddress[withdrawalAddress].length;
	for (j = 0; j < arrLength; j++) {
	  ...
	}

Description

The function withdrawTokens loops over an array for bookkeeping which in some cases can get lengthy enough to cause gas overflow issues. This will make the escrows get locked forever, and no account can withdraw it.

Furthermore, the gas usage can vary based on the length of the array, and the withdrawal account spends this extra gas.

Remediation

Consider using a different data structure that avoids unnecessary looping. OpenZeppelin's EnumerableSet is one such data structure that allows removing elements from an array without using loops.

Status: Fixed

This has been fixed in commit 6bfe159a79b6c756f86fe18571cb1dadcc4ba3eb.

4. Tokens with transfer fee are not supported

Description

While creating an escrow, the functions always expect the token to transfer the same amount as requested. Though this is not a common implementation, some tokens can charge a fee for each transfer resulting in fewer tokens received than what is requested.

If any such tokens are used to create escrows, any withdrawal will take the tokens locked for other users to complete the transaction.

Remediation
This is marked as a minor issue since it's not a very common implementation. Since the escrow allows the user to choose a token, consider updating the tokenAmount based on the number of tokens received instead of the value supplied by the user.

It is recommended to warn the user in the application or documentation if the team decides not to support such tokens.

Status: Acknowledged

Informational

5. Loss of precision while calculating vesting discount

Line	Code
142-145	uint256 fee = _amounts.length.mul(bnbFee).sub(_amounts.length.mul(bnbFee).mul(vestingDiscount).div(100));

Description
The vesting discount calculation in the method createMultipleLocks expects the result to not have any floating values. Since solidity does not support floating-point values out of the box, the vesting discount percentage calculation for very small fixed-point numbers would get rounded off to zero.

Remediation
Consider scaling the percentage value by some factor so that decimals are included. For example, use 5000 for 50.00%.

Status: Acknowledged

6. Supports all token contracts

Description
The smart contract allows the user to use any token contract, even custom created, for creating escrows. This, in most cases, is harmless, but a malicious user can use this contract as a medium for scams or other potential hacks/exploits.

Remediation

Consider warning the users of the potential side effects. If possible, consider using a whitelist to limit the tokens that can be used for escrow.

Status: Acknowledged

7. Violation of checks-effects-interaction pattern

Line	Code

262 - 263	if (remainingBnbFees > 0) {
	withdrawalAddress.transfer(remainingBnbFees);
	remainingBnbFees = 0;
	}

Description

The withdrawal method does not follow the checks-effects-interaction pattern. The method updates the state after an external interaction.

Remediation

Consider following the checks-effects-interaction pattern.

Status: Fixed

This has been fixed in commit be54d3810d7d0a5fc06d9426b4d5c9706c36610e.

8. Unused/unnecessary imports

```
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

Description

From solidity version 0.8.0, the compiler includes a built-in overflow check, and the SafeMath library is no longer required.
Reentrancy guard is not used in the smart contract, and the import can be removed.

Remediation

Consider removing the unused and unnecessary imports.

Status: Fixed

This has been fixed in commit be54d3810d7d0a5fc06d9426b4d5c9706c36610e.

9. Input address validation

Description

The methods lockTokens and createMultipleLocks do not validate the withdrawalAddress. Since an escrow cannot be changed after it is created, any accidental inputs can cause the funds to be locked forever.

Remediation

Consider validating the input address for address(0).

Status: Fixed

This has been fixed in commit be54d3810d7d0a5fc06d9426b4d5c9706c36610e.

10. Add revert messages

Description

The smart contract is missing revert messages for most of the require conditions. Adding revert messages can help identify the reverts easily.

Remediation

Consider adding messages to revert conditions for improved readability.

Status: Fixed

This has been fixed in commit be54d3810d7d0a5fc06d9426b4d5c9706c36610e.

11. Require statement with tautology

Line	Code
247	require(fee >= 0);

Description

The method setBnbFee checks whether the parameter fee is greater than or equal to zero. Since the type of fee is uint256, it is always greater than or equal to zero, and there is no need for redundant validation.

Remediation

Consider removing the redundant validation.

Status: Fixed

This has been fixed in commit be54d3810d7d0a5fc06d9426b4d5c9706c36610e.

Functional test

Function Names	Testing results
lockTokens	Passed
withdrawTokens	Passed
createMultipleLocks	Passed
getTotalTokenBalance	Passed
getTokenBalanceByAddress	Passed
getAllDepositIds	Passed
getDepositDetails	Passed
getDepositsByWithdrawalAddress	Passed
setBnbFee	Passed
setVestingDiscount	Passed
withdrawFees	Passed

Automated Testing

Slither

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

```
INFO:Detectors:
BPADVault.setBnbFee(uint256) (BPADVault.sol#246-249) contains a tautology or contradiction:
        - require(bool)(fee >= 0) (BPADVault.sol#247)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
BPADVault.withdrawFees(address).withdrawalAddress (BPADVault.sol#257) lacks a zero-check on :
                - withdrawalAddress.transfer(remainingBnbFees) (BPADVault.sol#262)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128) (BPADVault.sol#132-193) has external calls inside a loop: require
Vault.sol#178-184)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128) (BPADVault.sol#132-193):
        External calls:
        - require(bool)(IERC20(_tokenAddress).transferFrom(msg.sender,address(this),_amounts[i])) (BPADVault.sol#178-184)
        Event emitted after the call(s):
        - LogTokensLocked(_tokenAddress,msg.sender,_amounts[i],_unlockTimes[i],depositId) (BPADVault.sol#185-191)
Reentrancy in BPADVault.lockTokens(address,address,uint256,uint256,uint128) (BPADVault.sol#48-94):
        External calls:
        - require(bool)(IERC20(_tokenAddress).transferFrom(msg.sender,address(this),_amount)) (BPADVault.sol#80-86)
        Event emitted after the call(s):
        - LogTokensLocked(_tokenAddress,msg.sender,_amount,_unlockTime,depositId) (BPADVault.sol#87-93)
Reentrancy in BPADVault.withdrawTokens(uint256) (BPADVault.sol#96-130):
        External calls:
        - require(bool)(IERC20(tokenAddress).transfer(msg.sender,amount)) (BPADVault.sol#128)
        Event emitted after the call(s):
        - LogTokensWithdrawn(tokenAddress,withdrawalAddress,amount,_id) (BPADVault.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
BPADVault.lockTokens(address,address,uint256,uint256,uint128) (BPADVault.sol#48-94) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(_unlockTime > block.timestamp) (BPADVault.sol#57)
BPADVault.withdrawTokens(uint256) (BPADVault.sol#96-130) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp >= lockedToken[_id].unlockTime) (BPADVault.sol#97)
BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128) (BPADVault.sol#132-193) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(_unlockTimes[i] > block.timestamp) (BPADVault.sol#155)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.8.0', '^0.8.0']
        - 0.8.0 (BPADVault.sol#1)
        - ^0.8.0 (openzeppelin-contracts/contracts/access/Ownable.sol#3)
        - ^0.8.0 (openzeppelin-contracts/contracts/security/ReentrancyGuard.sol#3)
        - ^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#3)
        - ^0.8.0 (openzeppelin-contracts/contracts/utils/Context.sol#3)
        - ^0.8.0 (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

```
INFO:Detectors:
BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128) (BPADVault.sol#132-193) has costly operations inside a loop:
        - _id = ++ depositId (BPADVault.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
Context._msgData() (openzeppelin-contracts/contracts/utils/Context.sol#20-22) is never used and should be removed
SafeMath.div(uint256,uint256,string) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#190-199) is never used and should be removed
SafeMath.mod(uint256,uint256) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#150-152) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#216-225) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#167-176) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#21-27) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#63-68) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#75-80) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#46-56) is never used and should be removed
SafeMath.trySub(uint256,uint256) (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#34-39) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.0 (BPADVault.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (openzeppelin-contracts/contracts/access/Ownable.sol#3) necessitates a version too recent to be trusted. Consider deplo
Pragma version^0.8.0 (openzeppelin-contracts/contracts/security/ReentrancyGuard.sol#3) necessitates a version too recent to be trusted. Cons
Pragma version^0.8.0 (openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to be trusted. Consider d
Pragma version^0.8.0 (openzeppelin-contracts/contracts/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploy
Pragma version^0.8.0 (openzeppelin-contracts/contracts/utils/math/SafeMath.sol#3) necessitates a version too recent to be trusted. Consider
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter BPADVault.lockTokens(address,address,uint256,uint256,uint128)._tokenAddress (BPADVault.sol#49) is not in mixedCase
Parameter BPADVault.lockTokens(address,address,uint256,uint256,uint128)._withdrawalAddress (BPADVault.sol#50) is not in mixedCase
Parameter BPADVault.lockTokens(address,address,uint256,uint256,uint128)._amount (BPADVault.sol#51) is not in mixedCase
Parameter BPADVault.lockTokens(address,address,uint256,uint256,uint128)._unlockTime (BPADVault.sol#52) is not in mixedCase
Parameter BPADVault.lockTokens(address,address,uint256,uint256,uint128)._purpose (BPADVault.sol#53) is not in mixedCase
Parameter BPADVault.withdrawTokens(uint256)._id (BPADVault.sol#96) is not in mixedCase
Parameter BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128)._tokenAddress (BPADVault.sol#133) is not in mixedCase
Parameter BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128)._withdrawalAddress (BPADVault.sol#134) is not in mixedC
Parameter BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128)._amounts (BPADVault.sol#135) is not in mixedCase
Parameter BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128)._unlockTimes (BPADVault.sol#136) is not in mixedCase
Parameter BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128)._purpose (BPADVault.sol#137) is not in mixedCase
Parameter BPADVault.getTotalTokenBalance(address)._tokenAddress (BPADVault.sol#195) is not in mixedCase
Parameter BPADVault.getTokenBalanceByAddress(address,address)._tokenAddress (BPADVault.sol#204) is not in mixedCase
Parameter BPADVault.getTokenBalanceByAddress(address,address)._walletAddress (BPADVault.sol#205) is not in mixedCase
Parameter BPADVault.getDepositDetails(uint256)._id (BPADVault.sol#214) is not in mixedCase
Parameter BPADVault.getDepositsByWithdrawalAddress(address)._withdrawalAddress (BPADVault.sol#238) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in BPADVault.withdrawFees(address) (BPADVault.sol#257-265):
        External calls:
        - withdrawalAddress.transfer(remainingBnbFees) (BPADVault.sol#262)
        State variables written after the call(s):
        - remainingBnbFees = 0 (BPADVault.sol#263)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
        - require(bool)(_unlockTime < 10000000000) (BPADVault.sol#56)
BPADVault.createMultipleLocks(address,address,uint256[],uint256[],uint128) (BPADVault.sol#132-193) uses literals with too many digits:
        - require(bool)(_unlockTimes[i] < 10000000000) (BPADVault.sol#154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
withdrawTokens(uint256) should be declared external:
        - BPADVault.withdrawTokens(uint256) (BPADVault.sol#96-130)
getTotalTokenBalance(address) should be declared external:
        - BPADVault.getTotalTokenBalance(address) (BPADVault.sol#195-201)
getTokenBalanceByAddress(address,address) should be declared external:
        - BPADVault.getTokenBalanceByAddress(address,address) (BPADVault.sol#203-208)
getAllDepositIds() should be declared external:
        - BPADVault.getAllDepositIds() (BPADVault.sol#210-212)
getDepositDetails(uint256) should be declared external:
        - BPADVault.getDepositDetails(uint256) (BPADVault.sol#214-236)
getDepositsByWithdrawalAddress(address) should be declared external:
        - BPADVault.getDepositsByWithdrawalAddress(address) (BPADVault.sol#238-244)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (openzeppelin-contracts/contracts/access/Ownable.sol#53-55)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (openzeppelin-contracts/contracts/access/Ownable.sol#61-64)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Closing Summary

Overall, the smart contracts are decently written. No critical issues were found in the provided smart contract. There are several minor issues, and it is recommended to fix them before deploying to the main network. The smart contract supplied was missing test cases and build configurations that described the deployment and version configuration.

No reentrancy or Back-Door Entry was found in the contract, but relying on other contracts always comes with some risk.

Disclaimer

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

BLOCKPAD

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com