QuillAudits

Audit Report
September, 2021

For

BitBook

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the BitBook Staking 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 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 to 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 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 four 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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	1	0	3
Closed	1	3	3	4

Introduction

During the period of **September 16, 2021, to September 22, 2021** - QuillAudits Team performed a security audit for **BitBookStaking** smart contracts.

The code for the audit was taken from following the official link: https://github.com/AbdulRafaySiddiqui/bitbook-staking-contracts

Code is deployed on the below address: https://bscscan.com/address/0x8a5b84c34faa86af794f93c961dfc3ae4b75d633#code

Version Number	Date	Commit ID	Files
1	September 16	f2389be21b817ed16d668354050630f6d75d88cb	BitBookStaking.sol
2	September 23	dc09ae52cde37942257b05245ec8d705f1b116a2	BitBookStaking.sol

Issues Found

A. Contract – BitBookStaking.sol

High severity issues

1. **massUpdatePools() might be reverted**

Description

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully: Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can stall the complete contract at a certain point. Additionally, using unbounded loops can incur a lot of avoidable gas costs.

Therefore, the poolInfo.length in massUpdatePools() function might get to the maximum value which is set to 1000 in the future. During the course of the audit, the transaction costs around 30869 gas per transaction which means that if the poolInfo.length reaches 1000 (tested in Ropsten), hence, it would take approximately 30,869,000 gas in total.

However, the gas that can be sent to the massUpdatePools() function must be less than the block gas limit which is 30 Million gas, which could start failing the transaction because of insufficient gas, once it starts consuming more gas units than the block gas limit.

Remediation

Carefully test how many items at maximum you can pass to the function to make it successful, which is the maxLength variable in the below code.

One of solution we can provide is that to avoid the "out of gas" issue, we can pass in the number of iterations a loop can execute:

```
function massUpdatePools(uint256 from, uint256 to) public {
        uint256 length = poolInfo.length;
         require(from < length, “from should be less than the length”);
       require(to <= length, “to should be less than the length”);
       require( to.sub(from) < maxLength, “the interval has reached to the maximum length”)
         for (uint256 pid = from; pid < to; ++pid) {
             updatePool(pid);
         }
    }
```

The preceding code would help the caller of the contract create multiple batches to update the pool. This avoids the transaction failure issues related to insufficient gas.

Status: Fixed. The Auditee removed massUpdatePools() function in version 02.

Medium severity issues

2. Insufficient check in the canHarvest() function

```
function canHarvest(uint256 _pid, address _user) public view returns (bool) {
        UserInfo storage user = userInfo[_pid][_user];
        return block.timestamp >= user.nextHarvestUntil;
    }
```

Description

The canHarvest() always returns true even if the _pid and _user do not exist, which means that the function does not validate the _pid and _user variables. Therefore, the block.timestamp is always greater than user.nextHarvestUntil (whereby the nextHarvestUntil = 0, if the _user is invalid)

Remediation

We recommend validating and checking the existence of _pid and _user.

Status: Fixed. The Auditee added a check for user.amount in version 02.

3. Centralization Risks

Description
The role owner has the authority to update the critical settings:
- set()
- setLockDeposit()

Remediation
We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

- with reasonable latency for community awareness on privileged operations;
- Multisig with community-voted 3rd-party independent co-signers;
- DAO or Governance module increasing transparency and community involvement;

Status: Acknowledged

4. Lack of event emissions

Description
The missing event makes it difficult to track off-chain liquidity fee changes. An event should be emitted for significant transactions calling the following functions:
- set()
- depositRewardToken()
- emergencyAdminWithdraw()
- updatePaused()
- setLockDeposit()

Remediation
We recommend emitting an event to log the update of the following variables:
- _tokenPerBlock, _depositFeeBP, _minDeposit, _harvestInterval in set() function
- msg.sender, poolId, amount in depositRewardToken() function.
- _pid, balanceOf(address(this)), pool.accTokenPerShare, pool.tokenPerBlock, pool.lastRewardBlock in emergencyAdminWithdraw() function.
- paused in updatePaused() function.
- pid, poolInfo[pid].lockDeposit in setLockDeposit() function.

Status: Fixed. The event emissions for the aforementioned functions were added in version 02.

5. **Loops in the Contract are extremely costly**

Description
The for loops in the entire codebase includes state variables .length of a non-memory array, in the condition of the for loops. As a result, these state variables consume a lot more extra gas for every iteration of the for a loop.

Remediation
We recommend using a local variable instead of a state variable .length in a loop for the entire codebase. For example:

```
_withdrawFeeLength =_withdrawFee.length
for (uint i=0; i < _withdrawFeeLength; i++) {
.....
}
```

Status: Fixed in version 02.

Low level severity issues

6. Comparison to boolean constants

Description
We have found that boolean variables in the contract are compared to true or false, for example:
- require(paused == false, 'BITBOOK_STAKING: Paused!');

While those boolean variables can be used directly and do not need to be compared to true or false.

Remediation
We recommend removing the equality to the boolean constant in the entire codebase.

Status: Fixed in version 02.

7. Missing zero address validation

Description
We've detected missing zero address validation for the following parameters:
- _owner and token in constructor()
- _stakedToken and _rewardToken in add() function

Remediation
Consider implementing require statements where appropriate to validate all user-controlled input, including constructor, to avoid the potential for erroneous values to result in unexpected behaviors or wasted gas.

Status: Fixed in version 02.

8. Lack of Input Validation

Description

The balance of the msg.sender is not checked before withdrawal in the emergencyWithdraw() function. This issue also occurs in the emergencyAdminWithdraw() function, the balance of the address(this) is not checked against the empty balance

Remediation

We recommend adding an empty balance check for the emergencyAdminWithdraw() and emergencyAdminWithdraw() function, which helps to avoid the potential for erroneous values to result in unexpected behaviors or wasted gas.

Status: Fixed in version 02.

Informational Issues

9. State Variable Default Visibility

L63: mapping(address => mapping(address => bool)) poolExists;

Description
The Visibility of the aforementioned variable is not defined. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

The default is internal for state variables, but it should be made explicit.

Status: Fixed in version 02.

10. Not using delete to zero values

In the emergencyWithdraw() function within the contract, the following variables are manually replaced with Zero:
- user.amount = 0;
- user.rewardDebt = 0;
- user.rewardLockedUp = 0;
- user.nextHarvestUntil = 0;

And other places in emergencyAdminWithdraw() , such as:
- pool.accTokenPerShare = 0;
- pool.tokenPerBlock = 0;

To simplify the code and clarify intent, consider using delete instead.

Status: Fixed in version 02.

11. Use double quotes for string literals

Description
Single quote found in the above string variables. Whilst, the double quotes are being utilized for other string literals.

Remediation
We recommend using double quotes for string literals in the entire codebase

Status: Fixed in version 02.

12. Public function that could be declared external

Description
Using the external attribute for functions never called from the contract. Therefore, the following public functions that are never called by the contract should be declared external to save gas:
- initialize()
- set()
- massUpdatePools()
- deposit()
- withdraw()
- emergencyWithdraw()
- emergencyAdminWithdraw()
- updatePaused()
- setLockDeposit()

Status: Fixed in version 02.

13. Inconsistent coding style

Deviations from the Solidity Style Guide were identified throughout the entire codebase. Taking into consideration how much value a consistent coding style adds to the project's readability, enforcing a standard coding style with the help of linter tools such as Solhint is recommended.

Status: Acknowledged

14. block.timestamp may not be reliable

The Time contract uses the block.timestamp as part of the calculations and time checks.

Nevertheless, timestamps can be slightly altered by miners to favor them in contracts that have logics that depend strongly on them. Consider taking into account this issue and warning the users that such a scenario could happen.

Status: Acknowledged

15. Missing docstrings

It is extremely difficult to locate any contracts or functions, as they lack documentation. One consequence of this is that reviewers' understanding of the code's intention is impeded, which is significant because it is necessary to accurately determine both security and correctness.

They are additionally more readable and easier to maintain when wrapped in docstrings. The functions should be documented so that users can understand the purpose or intention of each function, as well as the situations in which it may fail, who is allowed to call it, what values it returns, and what events it emits.

Status: Acknowledged

Functional test

Function Names	Testing results
add()	Passed
deposit()	Passed
depositRewardToken()	Passed
emergencyAdminWithdrawn()	Passed
emergencyWithdraw()	Passed
initialize()	Passed
massUpdatepools()	Passed
renounceOwnership()	Passed
set()	Passed
setLockDeposit()	Passed
transferOwnership()	Passed
updatePaused()	Passed
updatePool()	Passed
withdrawn()	Passed
canHarvest()	Passed

Automated Tests

Slither

```
INFO:Detectors:
Reserve.safeTransfer(IBEP20,address,uint256) (BitBookStaking.sol#14-25) ignores return value by rewardToken.transfer(_to,tokenBal) (BitBookStaking.sol#21)
Reserve.safeTransfer(IBEP20,address,uint256) (BitBookStaking.sol#14-25) ignores return value by rewardToken.transfer(_to,_amount) (BitBookStaking.sol#23)
BitBookStaking.emergencyAdminWithdraw(uint256) (BitBookStaking.sol#320-327) ignores return value by pool.rewardToken.transfer(owner(),pool.rewardToken.balanceOf(address(this))) (BitBookStaking.sol#322)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
Reentrancy in BitBookStaking.deposit(uint256,uint256) (BitBookStaking.sol#227-253):
        External calls:
        - payOrLockupPendingToken(_pid) (BitBookStaking.sol#233)
                - rewardReserve.safeTransfer(pool.rewardToken,msg.sender,totalRewards) (BitBookStaking.sol#291)
        State variables written after the call(s):
        - user.depositTimestamp = block.timestamp (BitBookStaking.sol#236)
Reentrancy in BitBookStaking.deposit(uint256,uint256) (BitBookStaking.sol#227-253):
        External calls:
        - payOrLockupPendingToken(_pid) (BitBookStaking.sol#233)
                - rewardReserve.safeTransfer(pool.rewardToken,msg.sender,totalRewards) (BitBookStaking.sol#291)
        - pool.stakedToken.safeTransferFrom(address(msg.sender),address(this),_amount) (BitBookStaking.sol#238)
        - pool.stakedToken.safeTransfer(owner(),depositFee) (BitBookStaking.sol#243)
        State variables written after the call(s):
        - pool.stakedAmount = pool.stakedAmount.add(delta).sub(depositFee) (BitBookStaking.sol#245)
        - user.amount = user.amount.add(delta).sub(depositFee) (BitBookStaking.sol#244)
        - user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12) (BitBookStaking.sol#251)
Reentrancy in BitBookStaking.deposit(uint256,uint256) (BitBookStaking.sol#227-253):
        External calls:
        - payOrLockupPendingToken(_pid) (BitBookStaking.sol#233)
                - rewardReserve.safeTransfer(pool.rewardToken,msg.sender,totalRewards) (BitBookStaking.sol#291)
        - pool.stakedToken.safeTransferFrom(address(msg.sender),address(this),_amount) (BitBookStaking.sol#238)
        State variables written after the call(s):
        - pool.stakedAmount = pool.stakedAmount.add(delta) (BitBookStaking.sol#248)
        - user.amount = user.amount.add(delta) (BitBookStaking.sol#247)
Reentrancy in BitBookStaking.depositRewardToken(uint256,uint256) (BitBookStaking.sol#219-225):
        External calls:
        - _poolInfo.rewardToken.safeTransferFrom(msg.sender,address(rewardReserve),amount) (BitBookStaking.sol#222)
        State variables written after the call(s):
        - _poolInfo.rewardSupply += finalBalance.sub(initialBalance) (BitBookStaking.sol#224)
Reentrancy in BitBookStaking.withdraw(uint256,uint256) (BitBookStaking.sol#255-273):
        External calls:
        - payOrLockupPendingToken(_pid) (BitBookStaking.sol#260)
                - rewardReserve.safeTransfer(pool.rewardToken,msg.sender,totalRewards) (BitBookStaking.sol#291)
        State variables written after the call(s):
        - pool.stakedAmount = pool.stakedAmount.sub(_amount) (BitBookStaking.sol#267)
        - user.amount = user.amount.sub(_amount) (BitBookStaking.sol#263)
Reentrancy in BitBookStaking.withdraw(uint256,uint256) (BitBookStaking.sol#255-273):
        External calls:
        - payOrLockupPendingToken(_pid) (BitBookStaking.sol#260)
                - rewardReserve.safeTransfer(pool.rewardToken,msg.sender,totalRewards) (BitBookStaking.sol#291)
        - pool.stakedToken.safeTransfer(owner(),feeAmount) (BitBookStaking.sol#268)
        - pool.stakedToken.safeTransfer(address(msg.sender),amountToTransfer) (BitBookStaking.sol#269)
        State variables written after the call(s):
        - user.rewardDebt = user.amount.mul(pool.accTokenPerShare).div(1e12) (BitBookStaking.sol#271)
```

```
INFO:Detectors:
BitBookStaking.constructor(address,IBEP20,BitBookStaking.WithdrawFeeInterval[])._owner (BitBookStaking.sol#79) shadows:
        - Ownable._owner (@openzeppelin/contracts/access/Ownable.sol#19) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in BitBookStaking.deposit(uint256,uint256) (BitBookStaking.sol#227-253):
        External calls:
        - payOrLockupPendingToken(_pid) (BitBookStaking.sol#233)
                - rewardReserve.safeTransfer(pool.rewardToken,msg.sender,totalRewards) (BitBookStaking.sol#291)
        - pool.stakedToken.safeTransferFrom(address(msg.sender),address(this),_amount) (BitBookStaking.sol#238)
        - pool.stakedToken.safeTransfer(owner(),depositFee) (BitBookStaking.sol#243)
        Event emitted after the call(s):
        - Deposit(msg.sender,_pid,_amount) (BitBookStaking.sol#252)
Reentrancy in BitBookStaking.emergencyWithdraw(uint256) (BitBookStaking.sol#308-318):
        External calls:
        - pool.stakedToken.safeTransfer(address(msg.sender),amount) (BitBookStaking.sol#316)
        Event emitted after the call(s):
        - EmergencyWithdraw(msg.sender,_pid,amount) (BitBookStaking.sol#317)
Reentrancy in BitBookStaking.withdraw(uint256,uint256) (BitBookStaking.sol#255-273):
        External calls:
        - payOrLockupPendingToken(_pid) (BitBookStaking.sol#260)
                - rewardReserve.safeTransfer(pool.rewardToken,msg.sender,totalRewards) (BitBookStaking.sol#291)
        - pool.stakedToken.safeTransfer(owner(),feeAmount) (BitBookStaking.sol#268)
        - pool.stakedToken.safeTransfer(address(msg.sender),amountToTransfer) (BitBookStaking.sol#269)
        Event emitted after the call(s):
        - Withdraw(msg.sender,_pid,amountToTransfer) (BitBookStaking.sol#272)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
BitBookStaking.canHarvest(uint256,address) (BitBookStaking.sol#185-188) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= user.nextHarvestUntil (BitBookStaking.sol#187)
BitBookStaking.getWithdrawFee(uint256,uint256) (BitBookStaking.sol#299-306) uses timestamp for comparisons
        Dangerous comparisons:
        - depositTime <= _withdrawFee[i].day (BitBookStaking.sol#303)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (@openzeppelin/contracts/utils/Address.sol#26-35) uses assembly
        - INLINE ASM (@openzeppelin/contracts/utils/Address.sol#33)
Address._verifyCallResult(bool,bytes,string) (@openzeppelin/contracts/utils/Address.sol#171-188) uses assembly
        - INLINE ASM (@openzeppelin/contracts/utils/Address.sol#180-183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
BitBookStaking.deposit(uint256,uint256) (BitBookStaking.sol#227-253) compares to a boolean constant:
        -require(bool,string)(paused == false,BITBOOK_STAKING: Paused!) (BitBookStaking.sol#228)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['>=0.6.0<0.8.0', '>=0.6.2<0.8.0', '^0.6.0']
        - >=0.6.0<0.8.0 (@openzeppelin/contracts/access/Ownable.sol#3)
        - >=0.6.0<0.8.0 (@openzeppelin/contracts/math/SafeMath.sol#3)
        - >=0.6.2<0.8.0 (@openzeppelin/contracts/utils/Address.sol#3)
        - >=0.6.0<0.8.0 (@openzeppelin/contracts/utils/Context.sol#3)
        - >=0.6.0<0.8.0 (@openzeppelin/contracts/utils/ReentrancyGuard.sol#3)
        - ^0.6.0 (BitBookStaking.sol#3)
        - ABIEncoderV2 (BitBookStaking.sol#4)
        - ^0.6.0 (IBEP20.sol#3)
        - ^0.6.0 (IBEP20Mintable.sol#3)
        - ^0.6.0 (SafeBEP20.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (@openzeppelin/contracts/utils/Address.sol#79-81) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (@openzeppelin/contracts/utils/Address.sol#104-106) is never used and should be removed
Address.functionDelegateCall(address,bytes) (@openzeppelin/contracts/utils/Address.sol#153-155) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#163-169) is never used and should be removed
Address.functionStaticCall(address,bytes) (@openzeppelin/contracts/utils/Address.sol#129-131) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#139-145) is never used and should be removed
Address.sendValue(address,uint256) (@openzeppelin/contracts/utils/Address.sol#53-59) is never used and should be removed
Context._msgData() (@openzeppelin/contracts/utils/Context.sol#20-23) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (SafeBEP20.sol#46-60) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (SafeBEP20.sol#71-81) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (SafeBEP20.sol#62-69) is never used and should be removed
SafeMath.div(uint256,uint256,string) (@openzeppelin/contracts/math/SafeMath.sol#190-193) is never used and should be removed
SafeMath.mod(uint256,uint256) (@openzeppelin/contracts/math/SafeMath.sol#152-155) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (@openzeppelin/contracts/math/SafeMath.sol#210-213) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (@openzeppelin/contracts/math/SafeMath.sol#170-173) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (@openzeppelin/contracts/math/SafeMath.sol#24-28) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (@openzeppelin/contracts/math/SafeMath.sol#60-63) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (@openzeppelin/contracts/math/SafeMath.sol#70-73) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (@openzeppelin/contracts/math/SafeMath.sol#45-53) is never used and should be removed
SafeMath.trySub(uint256,uint256) (@openzeppelin/contracts/math/SafeMath.sol#35-38) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (@openzeppelin/contracts/access/Ownable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (@openzeppelin/contracts/math/SafeMath.sol#3) is too complex
Pragma version>=0.6.2<0.8.0 (@openzeppelin/contracts/utils/Address.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (@openzeppelin/contracts/utils/Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (@openzeppelin/contracts/utils/ReentrancyGuard.sol#3) is too complex
Pragma version^0.6.0 (BitBookStaking.sol#3) allows old versions
Pragma version^0.6.0 (IBEP20.sol#3) allows old versions
Pragma version^0.6.0 (IBEP20Mintable.sol#3) allows old versions
Pragma version^0.6.0 (SafeBEP20.sol#3) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (@openzeppelin/contracts/access/Ownable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (@openzeppelin/contracts/math/SafeMath.sol#3) is too complex
Pragma version>=0.6.2<0.8.0 (@openzeppelin/contracts/utils/Address.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (@openzeppelin/contracts/utils/Context.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (@openzeppelin/contracts/utils/ReentrancyGuard.sol#3) is too complex
Pragma version^0.6.0 (BitBookStaking.sol#3) allows old versions
Pragma version^0.6.0 (IBEP20.sol#3) allows old versions
Pragma version^0.6.0 (IBEP20Mintable.sol#3) allows old versions
Pragma version^0.6.0 (SafeBEP20.sol#3) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (@openzeppelin/contracts/utils/Address.sol#53-59):
        - (success) = recipient.call{value: amount}() (@openzeppelin/contracts/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (@openzeppelin/contracts/utils/Address.sol#114-121):
        - (success,returndata) = target.call{value: value}(data) (@openzeppelin/contracts/utils/Address.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#139-145):
        - (success,returndata) = target.staticcall(data) (@openzeppelin/contracts/utils/Address.sol#143)
Low level call in Address.functionDelegateCall(address,bytes,string) (@openzeppelin/contracts/utils/Address.sol#163-169):
        - (success,returndata) = target.delegatecall(data) (@openzeppelin/contracts/utils/Address.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
Parameter Reserve.safeTransfer(IBEP20,address,uint256)._to (BitBookStaking.sol#16) is not in mixedCase
Parameter Reserve.safeTransfer(IBEP20,address,uint256)._amount (BitBookStaking.sol#17) is not in mixedCase
Parameter BitBookStaking.add(uint256,IBEP20,IBEP20,uint16,uint256,uint256,BitBookStaking.WithdrawFeeInterval[])._tokenPerBlock (BitBookStaking.sol#114) is not in mixedCase
Parameter BitBookStaking.add(uint256,IBEP20,IBEP20,uint16,uint256,uint256,BitBookStaking.WithdrawFeeInterval[])._stakedToken (BitBookStaking.sol#115) is not in mixedCase
Parameter BitBookStaking.add(uint256,IBEP20,IBEP20,uint16,uint256,uint256,BitBookStaking.WithdrawFeeInterval[])._rewardToken (BitBookStaking.sol#116) is not in mixedCase
Parameter BitBookStaking.add(uint256,IBEP20,IBEP20,uint16,uint256,uint256,BitBookStaking.WithdrawFeeInterval[])._depositFeeBP (BitBookStaking.sol#117) is not in mixedCase
Parameter BitBookStaking.add(uint256,IBEP20,IBEP20,uint16,uint256,uint256,BitBookStaking.WithdrawFeeInterval[])._minDeposit (BitBookStaking.sol#118) is not in mixedCase
Parameter BitBookStaking.add(uint256,IBEP20,IBEP20,uint16,uint256,uint256,BitBookStaking.WithdrawFeeInterval[])._harvestInterval (BitBookStaking.sol#119) is not in mixedCase
Parameter BitBookStaking.set(uint256,uint256,uint16,uint256,uint256)._pid (BitBookStaking.sol#150) is not in mixedCase
Parameter BitBookStaking.set(uint256,uint256,uint16,uint256,uint256)._tokenPerBlock (BitBookStaking.sol#151) is not in mixedCase
Parameter BitBookStaking.set(uint256,uint256,uint16,uint256,uint256)._depositFeeBP (BitBookStaking.sol#152) is not in mixedCase
Parameter BitBookStaking.set(uint256,uint256,uint16,uint256,uint256)._minDeposit (BitBookStaking.sol#153) is not in mixedCase
Parameter BitBookStaking.set(uint256,uint256,uint16,uint256,uint256)._harvestInterval (BitBookStaking.sol#154) is not in mixedCase
Parameter BitBookStaking.getMultiplier(uint256,uint256)._from (BitBookStaking.sol#165) is not in mixedCase
Parameter BitBookStaking.getMultiplier(uint256,uint256)._to (BitBookStaking.sol#165) is not in mixedCase
Parameter BitBookStaking.pendingToken(uint256,address)._pid (BitBookStaking.sol#169) is not in mixedCase
Parameter BitBookStaking.pendingToken(uint256,address)._user (BitBookStaking.sol#169) is not in mixedCase
Parameter BitBookStaking.canHarvest(uint256,address)._pid (BitBookStaking.sol#185) is not in mixedCase
Parameter BitBookStaking.canHarvest(uint256,address)._user (BitBookStaking.sol#185) is not in mixedCase
Parameter BitBookStaking.updatePool(uint256)._pid (BitBookStaking.sol#197) is not in mixedCase
Parameter BitBookStaking.deposit(uint256,uint256)._pid (BitBookStaking.sol#227) is not in mixedCase
Parameter BitBookStaking.deposit(uint256,uint256)._amount (BitBookStaking.sol#227) is not in mixedCase
Parameter BitBookStaking.withdraw(uint256,uint256)._pid (BitBookStaking.sol#255) is not in mixedCase
Parameter BitBookStaking.withdraw(uint256,uint256)._amount (BitBookStaking.sol#255) is not in mixedCase
Parameter BitBookStaking.payOrLockupPendingToken(uint256)._pid (BitBookStaking.sol#275) is not in mixedCase
Parameter BitBookStaking.emergencyWithdraw(uint256)._pid (BitBookStaking.sol#308) is not in mixedCase
Parameter BitBookStaking.emergencyAdminWithdraw(uint256)._pid (BitBookStaking.sol#320) is not in mixedCase
Parameter BitBookStaking.updatePaused(bool)._value (BitBookStaking.sol#329) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (@openzeppelin/contracts/utils/Context.sol#21)" inContext (@openzeppelin/contracts/utils/Context.sol#15-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (@openzeppelin/contracts/access/Ownable.sol#54-57)
initialize() should be declared external:
        - BitBookStaking.initialize() (BitBookStaking.sol#94-103)
set(uint256,uint256,uint16,uint256,uint256) should be declared external:
        - BitBookStaking.set(uint256,uint256,uint16,uint256,uint256) (BitBookStaking.sol#149-163)
massUpdatePools() should be declared external:
        - BitBookStaking.massUpdatePools() (BitBookStaking.sol#190-195)
deposit(uint256,uint256) should be declared external:
        - BitBookStaking.deposit(uint256,uint256) (BitBookStaking.sol#227-253)
withdraw(uint256,uint256) should be declared external:
        - BitBookStaking.withdraw(uint256,uint256) (BitBookStaking.sol#255-273)
emergencyWithdraw(uint256) should be declared external:
        - BitBookStaking.emergencyWithdraw(uint256) (BitBookStaking.sol#308-318)
emergencyAdminWithdraw(uint256) should be declared external:
        - BitBookStaking.emergencyAdminWithdraw(uint256) (BitBookStaking.sol#320-327)
updatePaused(bool) should be declared external:
        - BitBookStaking.updatePaused(bool) (BitBookStaking.sol#329-331)
setLockDeposit(uint256,bool) should be declared external:
        - BitBookStaking.setLockDeposit(uint256,bool) (BitBookStaking.sol#333-335)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Mythril

```
enderphan@enderphan contracts_slither % myth a BitBookStaking.sol
The analysis was completed successfully. No issues were detected.
```

SOLHINT LINTER

```
BitBookStaking.sol
    3:1   error    Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver requirement  compiler-version
    6:8   error    Use double quotes for string literals                                   quotes
    7:8   error    Use double quotes for string literals                                   quotes
    8:8   error    Use double quotes for string literals                                   quotes
    9:8   error    Use double quotes for string literals                                   quotes
   10:8   error    Use double quotes for string literals                                   quotes
   11:8   error    Use double quotes for string literals                                   quotes
   63:5   warning  Explicitly mark visibility of state                                     state-visibility
   95:9   warning  Error message for require is too long                                   reason-string
   95:31  error    Use double quotes for string literals                                   quotes
  122:9   warning  Error message for require is too long                                   reason-string
  122:42  error    Use double quotes for string literals                                   quotes
  123:9   warning  Error message for require is too long                                   reason-string
  123:76  error    Use double quotes for string literals                                   quotes
  124:9   warning  Error message for require is too long                                   reason-string
  124:41  error    Use double quotes for string literals                                   quotes
  125:9   warning  Error message for require is too long                                   reason-string
  125:63  error    Use double quotes for string literals                                   quotes
  156:9   warning  Error message for require is too long                                   reason-string
  156:41  error    Use double quotes for string literals                                   quotes
  157:9   warning  Error message for require is too long                                   reason-string
  157:63  error    Use double quotes for string literals                                   quotes
  187:16  warning  Avoid to make time-based decisions in your business logic               not-rely-on-time
  228:34  error    Use double quotes for string literals                                   quotes
  230:36  error    Use double quotes for string literals                                   quotes
  235:13  warning  Error message for require is too long                                   reason-string
  235:59  error    Use double quotes for string literals                                   quotes
  236:37  warning  Avoid to make time-based decisions in your business logic               not-rely-on-time
  258:9   warning  Error message for require is too long                                   reason-string
  258:41  error    Use double quotes for string literals                                   quotes
  280:37  warning  Avoid to make time-based decisions in your business logic               not-rely-on-time
  289:41  warning  Avoid to make time-based decisions in your business logic               not-rely-on-time
  300:31  warning  Avoid to make time-based decisions in your business logic               not-rely-on-time
```

Results

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 BitBookStaking platform. We performed our audit according to the procedure described above.

The audit showed several high, medium, low, and informational severity issues. In the end, the majority of the issues were fixed and acknowledged by the Auditee.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or endorsement of the BitBookStaking 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 Bit book 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

BitBook

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com