ASTROBIATECH
BLOCKCHAIN SECURITY

MADE IN INDIA
BLOCKCHAIN SECURITY

SECURITY ASSESSMENT REPORT

PREPARED FOR

_patrickLpStaking

@astrobiatech

astrobiatech.in

TABLE OF CONTENTS

SCOPE OF AUDIT

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

CHECKED VULNERABILITIES

We have scanned the smart contract for commonly known and morespecific 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 & 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 thesmart contracts.

Static Analysis

Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to testsecurity 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 checkedand compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

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 movingto a live environment.

➢ MEDIUM 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 SEVERITY ISSUES

Low level severity issues can cause minor impact and or are just warningsthat can remain unfixed for now. It would be better to fix these issues atsome point in the future.

➢ INFORMATIONAL

These are severity four issues which indicate an improvement request, ageneral question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

ISSUES TABLE

TYPE	HIGH	MEDIUM	LOW	INFORMATIONAL
OPEN	1	0	0	0
ACKNOWLWDGENT	-	-	-	-
CLOSED	-	-	-	-

INTRODUCTION

On 01-11-2023 – Astrobiatech Blockchain Security Team performed security audit for _patrickLpStaking smart contract.

CONTRACT NAME	_patrickLpStaking
CONTRACT ADDRESS	0x82bf4533eeb922f463158a090cbfa81770a7cd14
BLOCKCHAIN	Ethereum

OVERVIEW

CONTRACT ADDRESS
0x82bf4533eeb922f463158a090cbfa81770a7cd14

CONTRACT NAME
_patrickLpStaking

CONTRACT CREATOR
0x56993A76e385aBf6A84B9Df9aF211E322B464E0F

OWNER ADDRESS
0x56993A76e385aBf6A84B9Df9aF211E322B464E0F

SOURCE CODE
Contract Source Code Verified at Ethereum Mainnet

OTHER SETTINGS
default evmVersion, MIT license

COMPILER VERSION
v0.8.0+commit.c7dfd78e

OPTIMIZATION ENABLED
No with 200 runs

Code is truncated to fit the constraints of this document.
https://etherscan.io/address/0x82bf4533eeb922f463158a090cbfa81770a7cd14#code

MANUAL ANALYSIS FINDINGS

HIGH

1. Uninitialized Reentrancy Guard

Description:-

In some cases, the Reentrancy Guard pattern may be incorrectly implemented, leading to an uninitialized or improperly initialized state. This can result in a false sense of security, as the contract could still be vulnerable to reentrancy attacks. Such an issue can arise when the Reentrancy Guard is not used consistently or if it's not set up correctly.

Recommendation:-

To address this issue, it's advisable to implement the Reentrancy Guard pattern. Utilize the built-in nonReentrant modifier provided by the ReentrancyGuard security library to secure these functions effectively. By ensuring consistent usage of the library and avoiding state manipulation that allows reentrancy attacks, you can enhance the security of the contract and protect against potential exploits.

AUTOMATED ANALYSIS

```
INFO:Detectors:
 _patrickLpStaking.erc20Transfer(address,uint256) (token.sol#1044-1047) ignores return value by erc20.transfer(_to,_amount) (token.sol#1045)
_patrickLpStaking.withdraw(uint256) (token.sol#1049-1051) ignores return value by erc20.transfer(msg.sender,_amount) (token.sol#1050)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

INFO:Detectors:
_patrickLpStaking.pending(uint256,address) (token.sol#949-963) performs a multiplication on the result of a division:
        - erc20Reward = nrOfBlocks.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (token.sol#958)
        - accERC20PerShare = accERC20PerShare.add(erc20Reward.mul(1e36).div(lpSupply)) (token.sol#959)
_patrickLpStaking.updatePool(uint256) (token.sol#984-1002) performs a multiplication on the result of a division:
        - erc20Reward = nrOfBlocks.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (token.sol#998)
        - pool.accERC20PerShare = pool.accERC20PerShare.add(erc20Reward.mul(1e36).div(lpSupply)) (token.sol#1000)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
_patrickLpStaking.updatePool(uint256) (token.sol#984-1002) uses a dangerous strict equality:
        - lpSupply == 0 (token.sol#992)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in _patrickLpStaking.deposit(uint256,uint256) (token.sol#1005-1017):
        External calls:
        - erc20Transfer(msg.sender,pendingAmount) (token.sol#1011)
                - erc20.transfer(_to,_amount) (token.sol#1045)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (token.sol#1013)
        State variables written after the call(s):
        - user.amount = user.amount.add(_amount) (token.sol#1014)
        _patrickLpStaking.userInfo (token.sol#882) can be used in cross function reentrancies:
        - _patrickLpStaking.deposit(uint256,uint256) (token.sol#1005-1017)
        - _patrickLpStaking.deposited(uint256,address) (token.sol#943-946)
        - _patrickLpStaking.emergencyWithdraw(uint256) (token.sol#1034-1041)
        - _patrickLpStaking.pending(uint256,address) (token.sol#949-963)
        - _patrickLpStaking.userInfo (token.sol#882)
        - _patrickLpStaking.withdraw(uint256,uint256) (token.sol#1020-1031)
        - user.rewardDebt = user.amount.mul(pool.accERC20PerShare).div(1e36) (token.sol#1015)
        _patrickLpStaking.userInfo (token.sol#882) can be used in cross function reentrancies:
        - _patrickLpStaking.deposit(uint256,uint256) (token.sol#1005-1017)
        - _patrickLpStaking.deposited(uint256,address) (token.sol#943-946)
        - _patrickLpStaking.emergencyWithdraw(uint256) (token.sol#1034-1041)
        - _patrickLpStaking.pending(uint256,address) (token.sol#949-963)
        - _patrickLpStaking.userInfo (token.sol#882)
        - _patrickLpStaking.withdraw(uint256,uint256) (token.sol#1020-1031)
Reentrancy in _patrickLpStaking.emergencyWithdraw(uint256) (token.sol#1034-1041):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),user.amount) (token.sol#1037)
        State variables written after the call(s):
        - user.amount = 0 (token.sol#1039)
        _patrickLpStaking.userInfo (token.sol#882) can be used in cross function reentrancies:
        - _patrickLpStaking.deposit(uint256,uint256) (token.sol#1005-1017)
        - _patrickLpStaking.deposited(uint256,address) (token.sol#943-946)
        - _patrickLpStaking.emergencyWithdraw(uint256) (token.sol#1034-1041)
        - _patrickLpStaking.pending(uint256,address) (token.sol#949-963)
        - _patrickLpStaking.userInfo (token.sol#882)
        - _patrickLpStaking.withdraw(uint256,uint256) (token.sol#1020-1031)
        - user.rewardDebt = 0 (token.sol#1040)
        _patrickLpStaking.userInfo (token.sol#882) can be used in cross function reentrancies:
        - _patrickLpStaking.deposit(uint256,uint256) (token.sol#1005-1017)
        - _patrickLpStaking.deposited(uint256,address) (token.sol#943-946)
        - _patrickLpStaking.emergencyWithdraw(uint256) (token.sol#1034-1041)
        - _patrickLpStaking.pending(uint256,address) (token.sol#949-963)
        - _patrickLpStaking.userInfo (token.sol#882)
        - _patrickLpStaking.withdraw(uint256,uint256) (token.sol#1020-1031)
Reentrancy in _patrickLpStaking.fund(uint256) (token.sol#908-913):
        External calls:
        - erc20.safeTransferFrom(address(msg.sender),address(this),_amount) (token.sol#911)
        State variables written after the call(s):
        - endBlock += _amount.div(rewardPerBlock) (token.sol#912)
        _patrickLpStaking.endBlock (token.sol#889) can be used in cross function reentrancies:
        - _patrickLpStaking.constructor(IERC20,uint256,uint256) (token.sol#895-900)
        - _patrickLpStaking.endBlock (token.sol#889)
        - _patrickLpStaking.fund(uint256) (token.sol#908-913)
        - _patrickLpStaking.pending(uint256,address) (token.sol#949-963)
        - _patrickLpStaking.totalPending() (token.sol#966-973)
        - _patrickLpStaking.updatePool(uint256) (token.sol#984-1002)
Reentrancy in _patrickLpStaking.withdraw(uint256,uint256) (token.sol#1020-1031):
        External calls:
        - erc20Transfer(msg.sender,pendingAmount) (token.sol#1026)
                - erc20.transfer(_to,_amount) (token.sol#1045)
        State variables written after the call(s):
        - user.amount = user.amount.sub(_amount) (token.sol#1027)
        _patrickLpStaking.userInfo (token.sol#882) can be used in cross function reentrancies:
        - _patrickLpStaking.deposit(uint256,uint256) (token.sol#1005-1017)
        - _patrickLpStaking.deposited(uint256,address) (token.sol#943-946)
        - _patrickLpStaking.emergencyWithdraw(uint256) (token.sol#1034-1041)
        - _patrickLpStaking.pending(uint256,address) (token.sol#949-963)
        - _patrickLpStaking.userInfo (token.sol#882)
        - _patrickLpStaking.withdraw(uint256,uint256) (token.sol#1020-1031)
        - user.rewardDebt = user.amount.mul(pool.accERC20PerShare).div(1e36) (token.sol#1028)
        _patrickLpStaking.userInfo (token.sol#882) can be used in cross function reentrancies:
        - _patrickLpStaking.deposit(uint256,uint256) (token.sol#1005-1017)
        - _patrickLpStaking.deposited(uint256,address) (token.sol#943-946)
        - _patrickLpStaking.emergencyWithdraw(uint256) (token.sol#1034-1041)
        - _patrickLpStaking.pending(uint256,address) (token.sol#949-963)
        - _patrickLpStaking.userInfo (token.sol#882)
        - _patrickLpStaking.withdraw(uint256,uint256) (token.sol#1020-1031)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

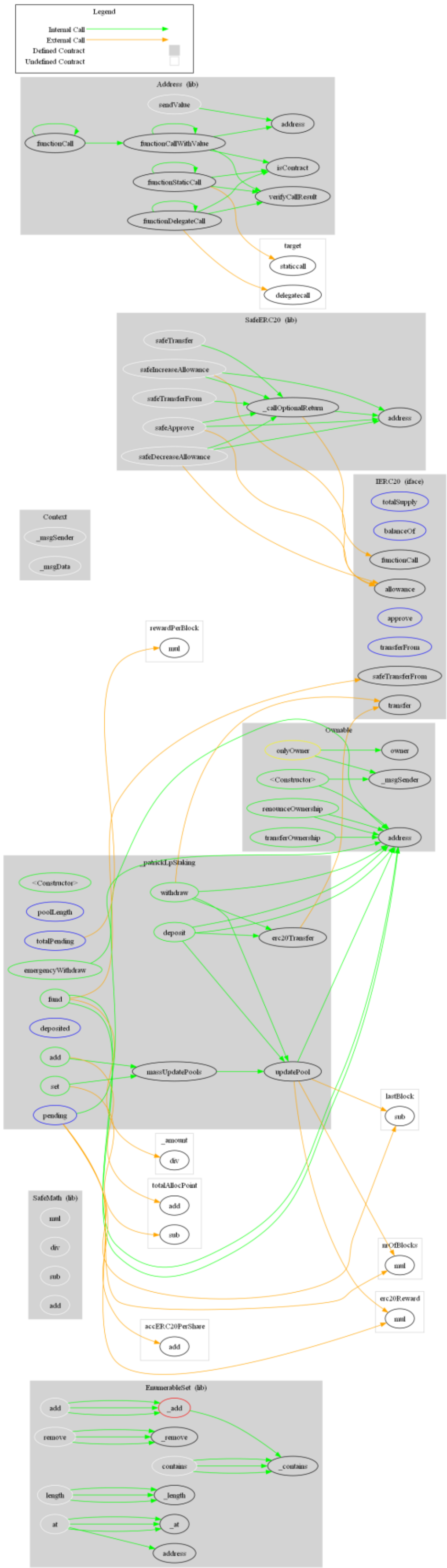
FUNCTIONAL ANALYSIS

```
|  Contract  |        Type        |       Bases      |                  |                 |
|:----------:|:-------------------:|:----------------:|:----------------:|:---------------:|
|     └      |  **Function Name**  |  **Visibility**  |  **Mutability**  |  **Modifiers**  |
||||||
| **_patrickLpStaking** | Implementation | Ownable |||
| └ | <Constructor> | Public ❗️ | 🔴 |NO ❗️ |
| └ | poolLength | External ❗️ |   |NO ❗️ |
| └ | fund | Public ❗️ | 🔴 |NO ❗️ |
| └ | add | Public ❗️ | 🔴 | onlyOwner |
| └ | set | Public ❗️ | 🔴 | onlyOwner |
| └ | deposited | External ❗️ |   |NO ❗️ |
| └ | pending | External ❗️ |   |NO ❗️ |
| └ | totalPending | External ❗️ |   |NO ❗️ |
| └ | massUpdatePools | Public ❗️ | 🔴 |NO ❗️ |
| └ | updatePool | Public ❗️ | 🔴 |NO ❗️ |
| └ | deposit | Public ❗️ | 🔴 |NO ❗️ |
| └ | withdraw | Public ❗️ | 🔴 |NO ❗️ |
| └ | emergencyWithdraw | Public ❗️ | 🔴 |NO ❗️ |
| └ | erc20Transfer | Internal 🔒 | 🔴 | |
| └ | withdraw | Public ❗️ | 🔴 | onlyOwner |


### Legend

|  Symbol  |  Meaning  |
|:--------:|-----------|
|    🔴    | Function can modify state |
|    💵    | Function is payable |
```

GRAPH TREE



SUMMARY

In this report, we have considered the security of the _patrickLpStaking smart contract. We performed our audit according to the procedure described above. 1 high severity were discovered during the audit.

DISCLAIMER

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constituteinvestment advice. No one shall have any right to rely on the report or its contents, and Astrobiatech Blockchain Security and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Astrobiatech BlockchainSecurity) owe no duty of care towards you or any other person, nor does Astrobiatech Blockchain Security make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Astrobiatech Blockchain Security hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effectin relation to the report. Except and only to the extent that it is prohibited by law, Astrobiatech Blockchain Security hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Astrobiatech Blockchain Security, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic lossor damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Proofed by Astrobiatech

ASTROBIATECH
BLOCKCHAIN SECURITY

https://astrobiatech.in

@astrobiatech