Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	Raku Doge Staking
Website:	rakudoge.rakucoin.com
Platform:	Ethereum
Language:	Solidity
Date:	July 21st, 2021

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

We were contracted by the Raku Doge team to perform the Security audit of the RAKUDOGE Staking smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on July 21st, 2021.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

Raku Doge is Raku Coin's Reward Eco-System token with a low supply to be used with Raku Vault for harvest or staking. Raku doge will have its own unique Tokenomics with a low supply that reward Holders for holding on to the Ecosystem Reward Token. Raku Coin Holders will be able to harvest or stake for Raku Doge based on a Supply/Demand Pool System.

Audit scope

Name	**Code Review and Security Analysis Report for Raku Doge Staking Smart Contract**
Platform	**Ethereum / Solidity**
File	Staking.sol
Smart Contract Online Code	https://etherscan.io/address/0x6908581000C01441a59afA664d39535e64928ae7#code
File MD5 Hash	8FD174A43107C2979163FCB21294243F
Audit Date	July 21st, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: Raku Doge Staking	**YES, This is valid.**
• The Owner can access functions like setRewardToken, setConversionRate, emergencyWithdraw, addPool, updatePool, removePool, etc.	**YES, This is valid. The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contract is **Secured**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.

We found 0 critical, 0 high, 0 medium and 1 low and some very low level technical issues.

Email: audit@EtherAuthority.io

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 1 smart contract. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. This is a compact and well written contract.

The libraries in the Raku Doge Staking are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Raku Doge Staking.

The Raku Doge Staking team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are **not well** commented on smart contracts.

Documentation

We were given Raku Doge Staking smart contract code in the form of an Etherscan web link. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are **not well** commented. So it is difficult to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website https://rakudoge.rakucoin.com/ which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects. And their core code blocks are written well.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

(1) Interface

(a) IERC20

(2) Inherited contracts

(a) Context

(b) Ownable

(3) Usages

(a) using SafeMath for uint256;

(4) Struct

(a) StakeInfo

(b) PoolInfo

(5) Events

(a) event Deposit(address indexed user,address indexed stakeToken,uint256 amount);

(b) event Withdraw(address indexed user,address indexed stakeToken,uint256 amount);

(c) event Claim(address indexed user, address indexed rewardToken,uint256 amount);

(d) event OwnershipTransferred(address indexed previousOwner,address indexed newOwner);

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	setRewardToken	write	Missing zero address validation	Refer Audit Findings
2	setConversionRate	write	Missing Events emitting	Refer Audit Findings
3	emergencyWithdraw	external	Missing Events emitting	Refer Audit Findings
4	addPool	external	Missing Events emitting	Refer Audit Findings
5	updatePool	external	Missing Events emitting	Refer Audit Findings

6	removePool	external	Missing Events emitting	Refer Audit Findings
7	deposit	external	Passed	No Issue
8	withdraw	external	Passed	No Issue
9	getRewardAmount	read	Passed	No Issue
10	getRemainingRewards	read	Passed	No Issue
11	getPoolCount	read	Passed	No Issue
12	getUserStakeCount	xread	Passed	No Issue
13	owner	read	Passed	No Issue
14	onlyOwner	modifier	Passed	No Issue
15	renounceOwnership	write	access only Owner	No Issue
16	transferOwnership	write	access only Owner	No Issue
17	_msgSender	internal	Passed	No Issue
18	_msgData	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Function input parameters lack of check:

```
function setRewardToken(address _rewardToken) public onlyOwner {
    rewardToken = _rewardToken;
}
```

Detects missing zero address validation to prevent contract loses, in function setRewardToken check the address is not zero_rewardToken.

Resolution: Check that the address is not zero.

Very Low / Discussion / Best practices:

(1) Missing Events emitting:

Many functions change state variables to emit events for these functions.

Functions needs events are :

- setRewardToken
- setConversionRate
- emergencyWithdraw
- addPool
- updatePool
- removePool

Resolution: For all the state variables which change runtime this needs to emit an event. Recommended to emit events for all these functions.

(2) Use the latest solidity version:

```
pragma solidity >=0.6.0 <0.8.0;
```

Using the latest solidity will prevent any compiler level bugs.

Resolution: Please use 0.8.6 which is the latest version.

.

Centralization

This smart contract has some functions which can be executed by Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- setRewardToken: The Owner can set a reward token.
- setConversionRate: The Owner can set a conversion rate.
- emergencyWithdraw: The Owner can check the reward token enough balance or not and check Emergency withdrawal balance or not.
- addPool: The Owner can add a new pool.
- updatePool: The Owner can update pool information.
- removePool: The Owner can check if the pool exists or not, and if it exists then remove the pool.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those are fixed/acknowledged in the smart contracts. **So it is good to go for the production.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **" Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

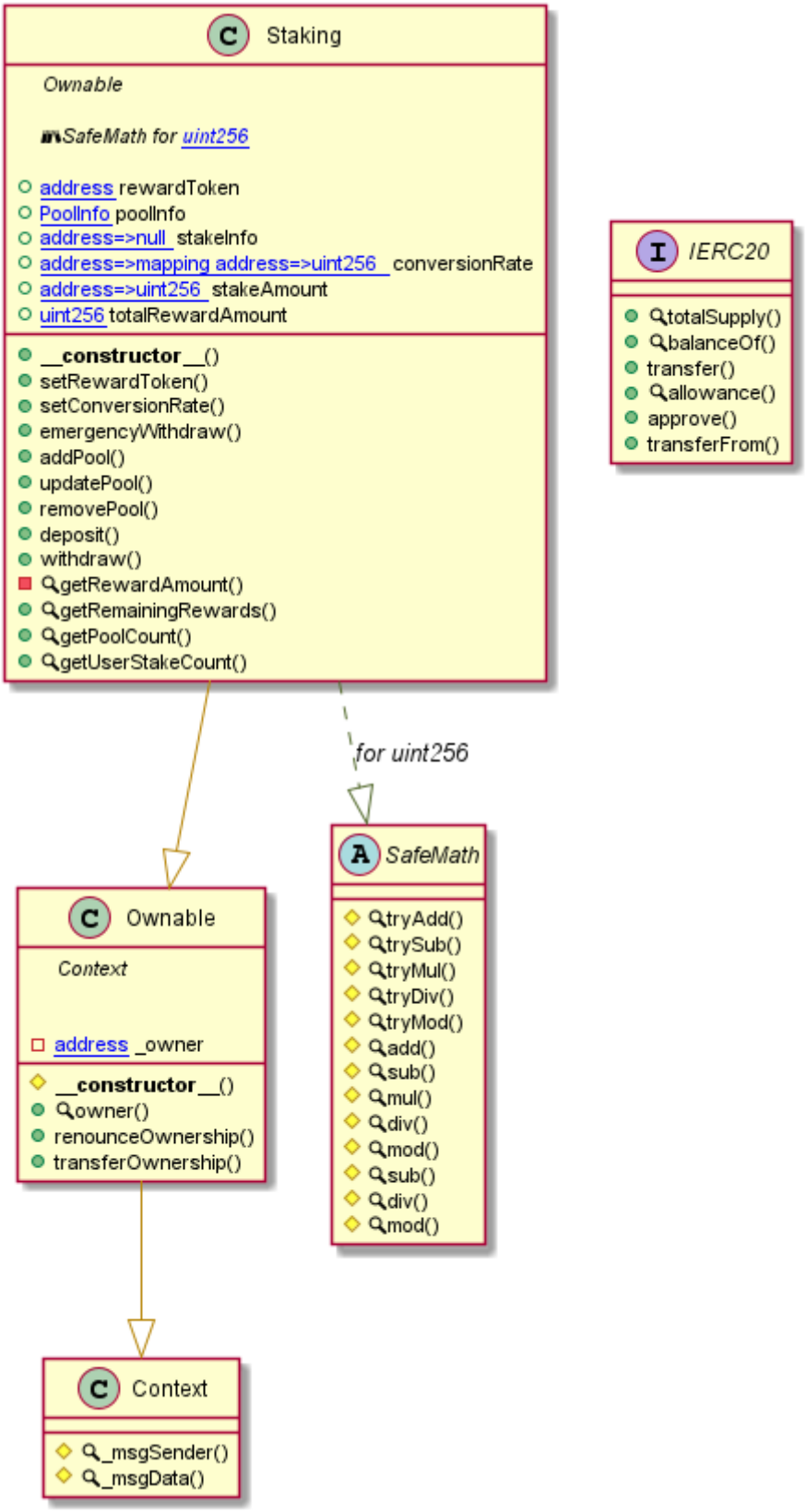
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Raku Doge Staking



Email: audit@EtherAuthority.io

Slither Results Log

Slither log >> Staking.sol

```
INFO:Detectors:
Reentrancy in Staking.deposit(uint256,uint256) (Staking.sol#526-558):
	External calls:
	- require(bool)(IERC20(poolInfo[pid].stakeToken).transferFrom(msg.sender,address(this),amount))
(Staking.sol#537-543)
	State variables written after the call(s):
	- stakeAmount[pool.stakeToken] = stakeAmount[pool.stakeToken].add(amount) (Staking.sol#546)
	- totalRewardAmount = totalRewardAmount.add(rewardAmount) (Staking.sol#555)
Reentrancy in Staking.withdraw(uint256) (Staking.sol#560-596):
	External calls:
	- require(bool,string)(IERC20(info.stakeToken).transfer(msg.sender,info.amount),Withdrawl failed)
(Staking.sol#568-571)
	- require(bool,string)(IERC20(rewardToken).transfer(msg.sender,info.rewardAmount),Claim failed)
(Staking.sol#583-586)
	State variables written after the call(s):
	- stakeInfo[msg.sender][stakeId] = stakeInfo[msg.sender][stakeInfo[msg.sender].length - 1]
(Staking.sol#592-594)
	- stakeInfo[msg.sender].pop() (Staking.sol#595)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Staking.setRewardToken(address)._rewardToken (Staking.sol#466) lacks a zero-check on :
		- rewardToken = _rewardToken (Staking.sol#467)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Staking.deposit(uint256,uint256) (Staking.sol#526-558):
	External calls:
	- require(bool)(IERC20(poolInfo[pid].stakeToken).transferFrom(msg.sender,address(this),amount))
(Staking.sol#537-543)
	State variables written after the call(s):
	-
stakeInfo[msg.sender].push(StakeInfo(pool.stakeToken,amount,rewardAmount,block.timestamp.add(pool.loc
kupDuration))) (Staking.sol#547-554)
Reentrancy in Staking.withdraw(uint256) (Staking.sol#560-596):
	External calls:
	- require(bool,string)(IERC20(info.stakeToken).transfer(msg.sender,info.amount),Withdrawl failed)
(Staking.sol#568-571)
	State variables written after the call(s):
	- stakeAmount[info.stakeToken] = stakeAmount[info.stakeToken].sub(info.amount)
(Staking.sol#574-576)
Reentrancy in Staking.withdraw(uint256) (Staking.sol#560-596):
	External calls:
	- require(bool,string)(IERC20(info.stakeToken).transfer(msg.sender,info.amount),Withdrawl failed)
(Staking.sol#568-571)
	- require(bool,string)(IERC20(rewardToken).transfer(msg.sender,info.rewardAmount),Claim failed)
(Staking.sol#583-586)
	State variables written after the call(s):
	- totalRewardAmount = totalRewardAmount.sub(info.rewardAmount) (Staking.sol#587)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Staking.deposit(uint256,uint256) (Staking.sol#526-558):
	External calls:
	- require(bool)(IERC20(poolInfo[pid].stakeToken).transferFrom(msg.sender,address(this),amount))
(Staking.sol#537-543)
	Event emitted after the call(s):
	- Deposit(msg.sender,pool.stakeToken,amount) (Staking.sol#557)
Reentrancy in Staking.withdraw(uint256) (Staking.sol#560-596):
	External calls:
```

- require(bool,string)(IERC20(info.stakeToken).transfer(msg.sender,info.amount),Withdrawl failed) (Staking.sol#568-571)
Event emitted after the call(s):
- Withdraw(msg.sender,info.stakeToken,info.amount) (Staking.sol#572)
Reentrancy in Staking.withdraw(uint256) (Staking.sol#560-596):
External calls:
- require(bool,string)(IERC20(info.stakeToken).transfer(msg.sender,info.amount),Withdrawl failed) (Staking.sol#568-571)
- require(bool,string)(IERC20(rewardToken).transfer(msg.sender,info.rewardAmount),Claim failed) (Staking.sol#583-586)
Event emitted after the call(s):
- Claim(msg.sender,rewardToken,info.rewardAmount) (Staking.sol#588)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Staking.withdraw(uint256) (Staking.sol#560-596) uses timestamp for comparisons
Dangerous comparisons:
- info.endTime < block.timestamp (Staking.sol#578)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Context._msgData() (Staking.sol#23-26) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Staking.sol#398-405) is never used and should be removed
SafeMath.mod(uint256,uint256) (Staking.sol#356-359) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Staking.sol#422-429) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (Staking.sol#374-381) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (Staking.sol#208-216) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (Staking.sol#256-263) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (Staking.sol#270-277) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (Staking.sol#237-249) is never used and should be removed
SafeMath.trySub(uint256,uint256) (Staking.sol#223-230) 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 (Staking.sol#6) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter Staking.setRewardToken(address)._rewardToken (Staking.sol#466) is not in mixedCase
Parameter Staking.setConversionRate(address,address,uint256)._conversionRate (Staking.sol#473) is not in mixedCase
Parameter Staking.addPool(address,uint256,uint256)._stakeToken (Staking.sol#496) is not in mixedCase
Parameter Staking.addPool(address,uint256,uint256)._rewardRate (Staking.sol#497) is not in mixedCase
Parameter Staking.addPool(address,uint256,uint256)._lockupDuration (Staking.sol#498) is not in mixedCase
Parameter Staking.updatePool(uint256,uint256,uint256)._rewardRate (Staking.sol#511) is not in mixedCase
Parameter Staking.updatePool(uint256,uint256,uint256)._lockupDuration (Staking.sol#512) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Staking.sol#24)" inContext (Staking.sol#18-27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (Staking.sol#79-82)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (Staking.sol#88-95)
setConversionRate(address,address,uint256) should be declared external:
- Staking.setConversionRate(address,address,uint256) (Staking.sol#470-476)
getPoolCount() should be declared external:
- Staking.getPoolCount() (Staking.sol#620-622)
getUserStakeCount(address) should be declared external:
- Staking.getUserStakeCount(address) (Staking.sol#624-626)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Staking.sol analyzed (5 contracts with 75 detectors), 34 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Email: audit@EtherAuthority.io