RD
AUDITORS

DGI Game Contract Code Review and Security Analysis Report

Customer: DGI Game
Prepared on: 12th February 2024
Platform: Ethereum
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of DGI Game
Platform	Ethereum/ Solidity
File	DGIStaking.sol
MD5 hash	ddcd573f7939783f838791a440247126
SHA256 hash	52e5a40f69185bea6c3b44a1804e7163843e300355792cba30d7e3985c663e96
File	SafeMath.sol
MD5 hash	d8601ab024d98063d1884414caa798c1
SHA256 hash	8213cd58437a8a6b5acb2a85358cd245f5ae0e44674af84c60a312b8b86049d7
File	SafeERC20.sol
MD5 hash	533fc0be719f87bef290bc6f6e3d7366
SHA256 hash	0f93551f327a6465365c72deff5a4a94c5d457bdd6484c9cf313a08bc4dc8506
File	ReentrancyGuard.sol

MD5 hash	ba1ddba253c8d2d51b645e806d3b86b9
SHA256 hash	4b3264a4e65f23fe5e65141a9dcbad21e7bdce9e1d4de44cb49bf0b4460caffc
File	Ownable.sol
MD5 hash	580e34fed6b52adce60e3a64311fe1ba
SHA256 hash	96a3b09372173d7174fcb0080a97c0cd9abb51cd31e71ecd597d62e0942cb7c4
File	IERC20.sol
MD5 hash	d41d8cd98f00b204e9800998ecf8427e
SHA256 hash	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
File	draftERC20Permit.sol
MD5 hash	91ff4c7f62df1f5d8aa65a61b335ab45
SHA256 hash	831d97e6913d2d8f540aa0f2f659c0c003145bafd2243e480f31e473a6046503
File	Context.sol
MD5 hash	c4b296fb9a98a645ca52cc72c3fbae06

SHA256 hash	6de5302543723d32c8eaf17becc4525936e16d9c4551455c93d306b9b72c0799
File	Address.sol
MD5 hash	d41d8cd98f00b204e9800998ecf8427e
SHA256 hash	e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
Date	12/02/2024

Introduction

RD Auditors (Consultant) were contracted by DGI Game (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer's smart contract and its code review conducted between 6th - 12th February 2024.

This contract consists of nine files.

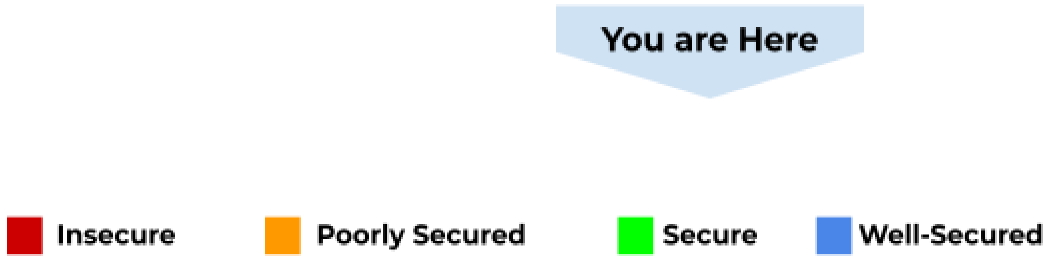
Project Scope

The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer's solidity smart contract is now **Secured.**



Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	0
🟥 Critical	0
🟧 High	0
🟨 Medium	0
🟩 Low	0
🟦 Very Low	0

Project Overview

The DGI Staking Contract stands at the pinnacle of token staking innovation, offering stakeholders unparalleled opportunities to strategically deploy their DGI tokens and unlock substantial rewards over time. This comprises flexible withdrawal options, sophisticated tracking systems, structured reward distributions, ensuring precision predictability and efficiency in the staking process. Additionally, incentivizing long-term commitment and fostering a culture of sustained engagement within the ecosystem.

Code Quality

The libraries within this smart contract are part of a 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 to a specific address and its properties/methods can be reused many times by other contracts.

The DGI Game team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the DGI game code as a link

https://etherscan.io/address/0x0c5901Bb3dFb0947566e7D4517841991fBD6bD87#code

The hash of that file is mentioned in the table. As mentioned above, it's commented on smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

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

AS-IS Overview

DGI Game.sol

File And Function Level Report

Contract:	DGIStaking
Import:	IERC20.sol, SafeERC20, ReentrancyGuard.sol, Ownable, SafeMath
Inherit:	ReentrancyGuard, Ownable
Observation:	Passed
Test Report:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	balanceOf	read	Passed	All Passed	No Issue	Passed
2	lastTimeRewardApplicable	read	Passed	All Passed	No Issue	Passed
3	RewardPerToken	read	Passed	All Passed	No Issue	Passed
	earned	read	Passed	All Passed	No Issue	Passed
4	getRewardForDuration	read	Passed	All Passed	No Issue	Passed
5	Stake	write	Passed	All Passed	No Issue	Passed
6	withdraw	write	Passed	All Passed	No Issue	Passed
7	getReward	write	Passed	All Passed	No Issue	Passed
8	exit	write	Passed	All Passed	No Issue	Passed
9	notifyRewardAmount	onlyOwner	Passed	All Passed	No Issue	Passed
10	migrateStaking	onlhOwner	Passed	All Passed	No Issue	Passed

11	recoverERC20	onlyOwner	Passed	All Passed	No Issue	Passed
12	SetRewardDuratin	onlyOwner	Passed	All Passed	No Issue	Passed

Library:	SafeMath
Observation:	Passed
Test Report:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	tryAdd	internal	Passed	All Passed	No Issue	Passed
2	trySub	internal	Passed	All Passed	No Issue	Passed
3	tryMul	internal	Passed	All Passed	No Issue	Passed
4	tryDiv	internal	Passed	All Passed	No Issue	Passed
5	tryMod	internal	Passed	All Passed	No Issue	Passed
6	Add	internal	Passed	All Passed	No Issue	Passed
7	Sub	internal	Passed	All Passed	No Issue	Passed
8	mul	internal	Passed	All Passed	No Issue	Passed
9	div	internal	Passed	All Passed	No Issue	Passed
10	mod	internal	Passed	All Passed	No Issue	Passed
11	sub	internal	Passed	All Passed	No Issue	Passed
12	div	internal	Passed	All Passed	No Issue	Passed
13	mod	internal	Passed	All Passed	No Issue	Passed

Library: SafeERC20

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	SafeTransfer	internal	Passed	All Passed	No Issue	Passed
2	SafeTransferFrom	internal	Passed	All Passed	No Issue	Passed
3	SafeApprove	internal	Passed	All Passed	No Issue	Passed
4	SafeIncreaseAllowance	internal	Passed	All Passed	No Issue	Passed
5	SafeDecrease Allowance	internal	Passed	All Passed	No Issue	Passed
6	SafePermit	internal	Passed	All Passed	No Issue	Passed
7	_CallOptional Return	write	Passed	All Passed	No Issue	Passed

Abstract: ReentrancyGuard

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_nonReentrantBefore	write	Passed	All Passed	No Issue	Passed
2	_nonReentrantAfter	write	Passed	All Passed	No Issue	Passed

Abstract: Ownable

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Owner	read	Passed	All Passed	No Issue	Passed
2	_checkOwner	internal	Passed	All Passed	No Issue	Passed
3	renounceOwnership	onlyOwner	Passed	All Passed	No Issue	Passed
4	transferOwnership	onlyOwner	Passed	All Passed	No Issue	Passed
5	_transferOwnership	internal	Passed	All Passed	No Issue	Passed

Interface: IERC20

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	totalSupply	read	Passed	All Passed	No Issue	Passed
2	balanceOf	read	Passed	All Passed	No Issue	Passed
3	transfer	external	Passed	All Passed	No Issue	Passed
4	allowance	external	Passed	All Passed	No Issue	Passed
5	approve	external	Passed	All Passed	No Issue	Passed
6	transferFrom	external	Passed	All Passed	No Issue	Passed

Interface: ERC20Permit

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Permit	read	Passed	All Passed	No Issue	Passed
2	nonces	read	Passed	All Passed	No Issue	Passed
3	DOMAIN_SEPARATOR	read	Passed	All Passed	No Issue	Passed

Abstract: context

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_msgSender	internal	Passed	All Passed	No Issue	Passed
2	_msgData	internal	Passed	All Passed	No Issue	Passed

library: Address

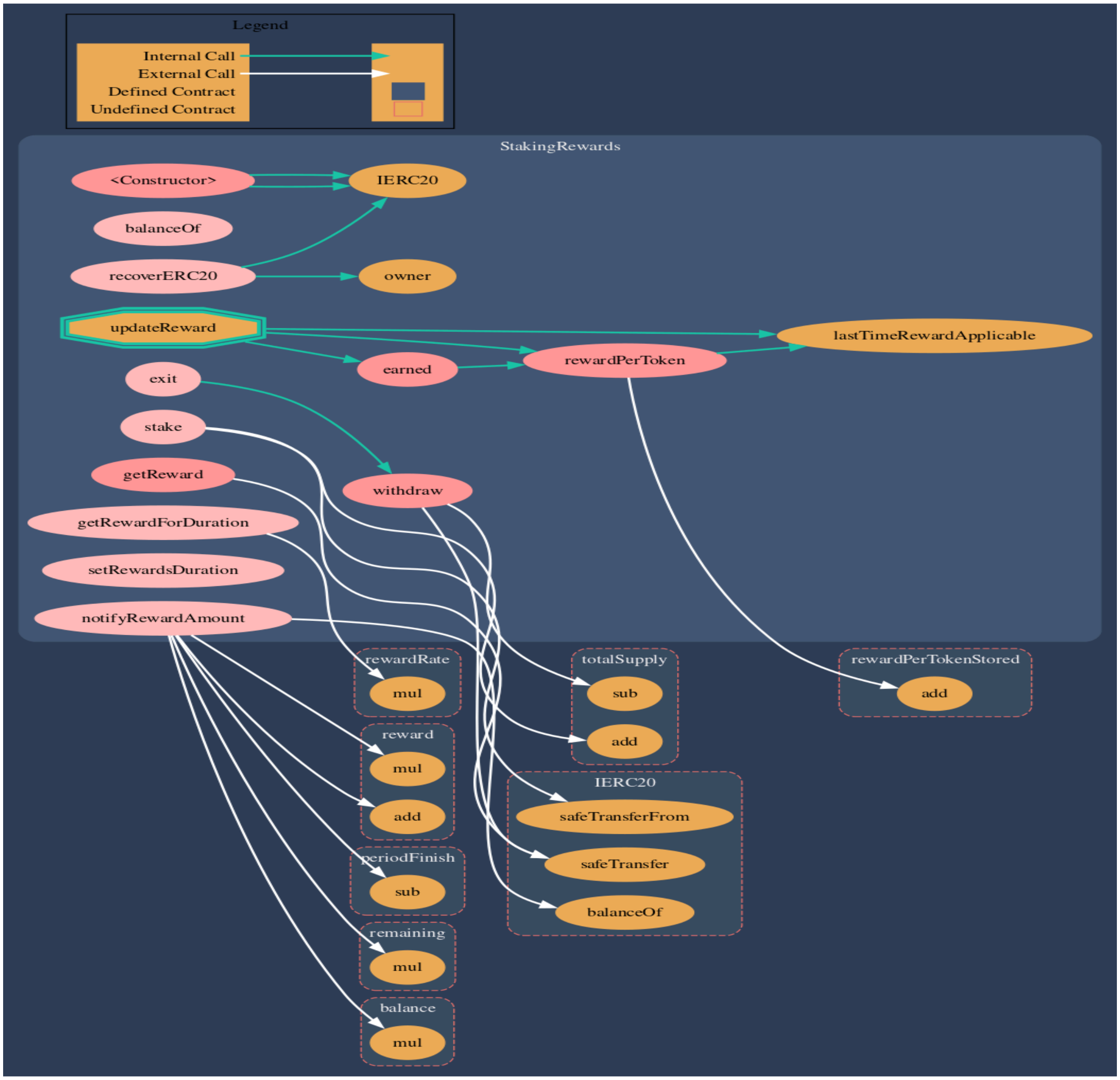
Observation: Passed

Test Report: Passed

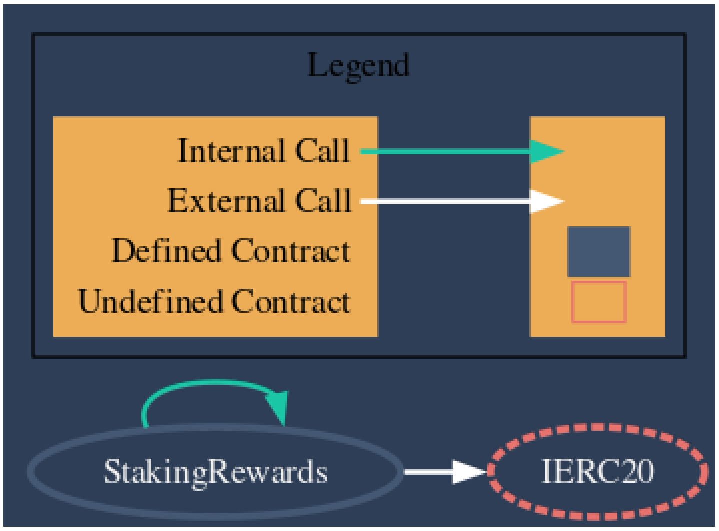
Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	isContract	internal	Passed	All Passed	No Issue	Passed
2	SendValue	internal	Passed	All Passed	No Issue	Passed
3	functionCall	internal	Passed	All Passed	No Issue	Passed
4	functionCall	internal	Passed	All Passed	No Issue	Passed

5	functionCallWithValue	internal	Passed	All Passed	No Issue	Passed
6	functionCallWithValue	internal	Passed	All Passed	No Issue	Passed
7	functionStaticCall	internal	Passed	All Passed	No Issue	Passed
8	functionStaticCall	internal	Passed	All Passed	No Issue	Passed
9	functionDelegateCall	internal	Passed	All Passed	No Issue	Passed
10	functionDelegateCall	internal	Passed	All Passed	No Issue	Passed
11	VerifyCallResultFromTarget	internal	Passed	All Passed	No Issue	Passed
12	VerifyCallResult	internal	Passed	All Passed	No Issue	Passed
13	_revert	read	Passed	All Passed	No Issue	Passed

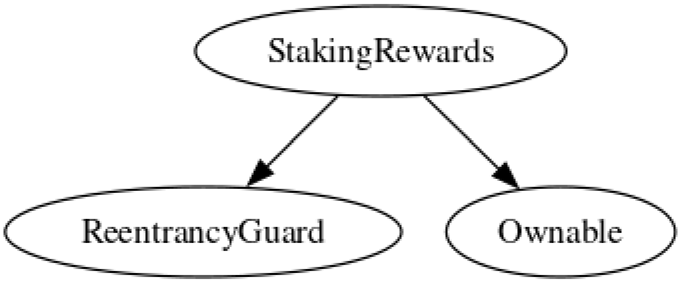
Code Flow Diagram - DGI Game



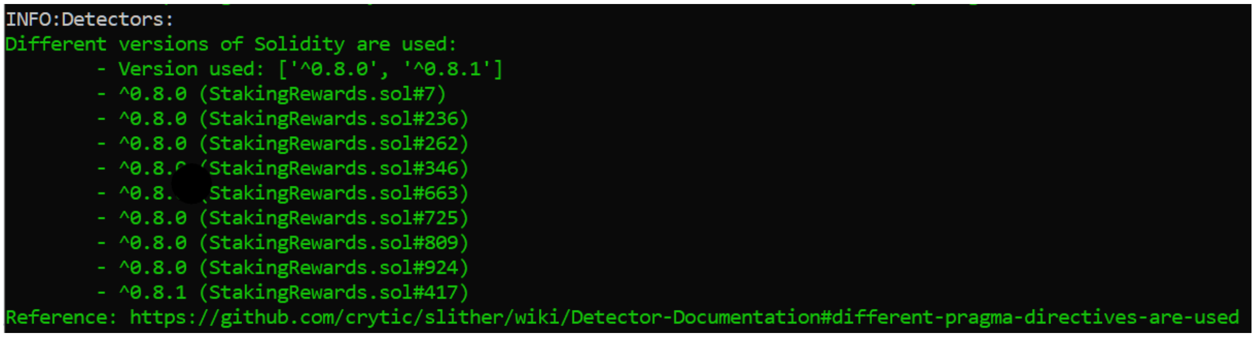
Interaction Diagram



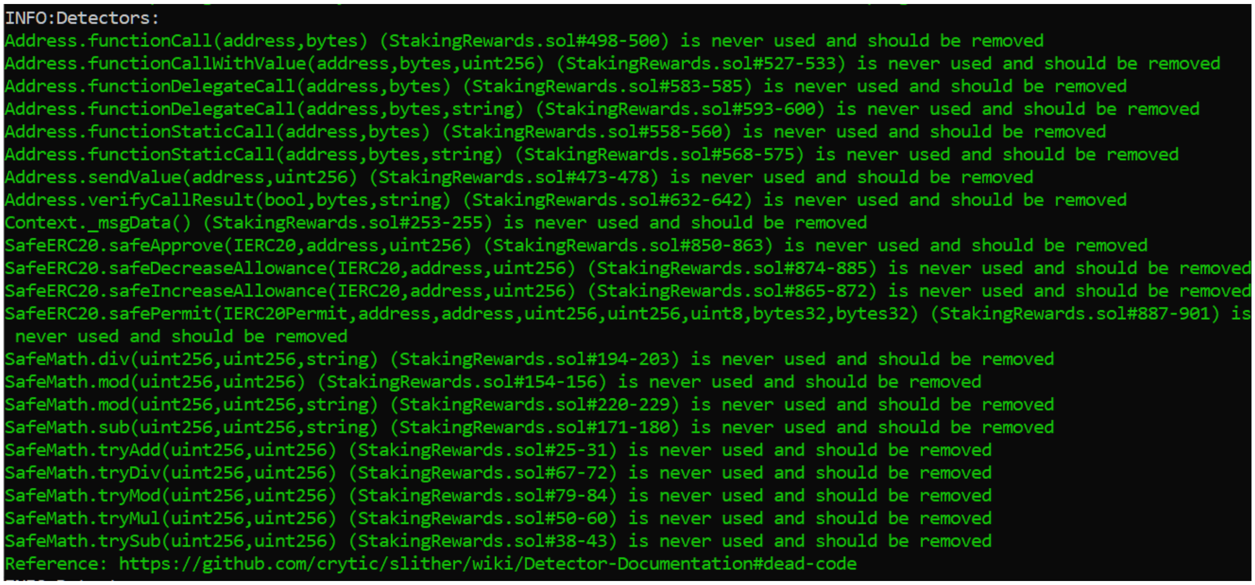
Inheritance Diagram



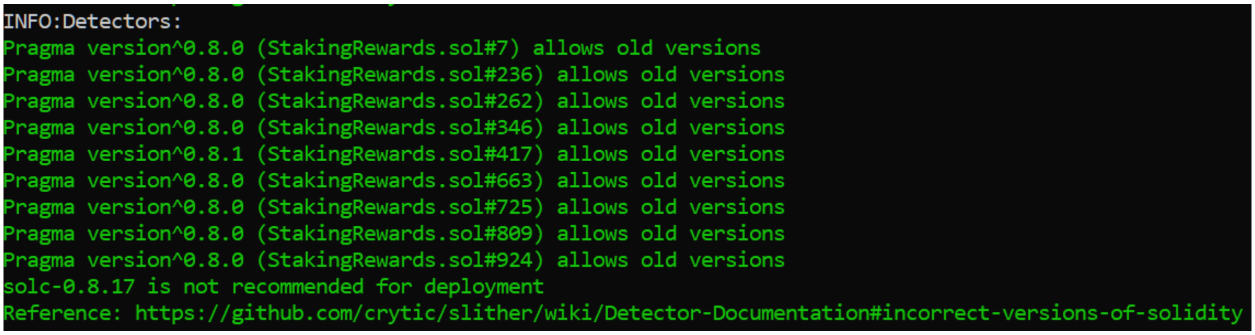
Code Flow Diagram - Slither Results Log



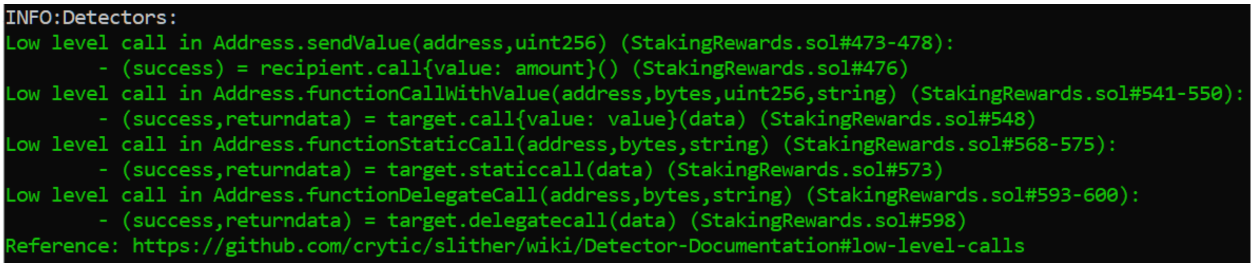
```
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1']
        - ^0.8.0 (StakingRewards.sol#7)
        - ^0.8.0 (StakingRewards.sol#236)
        - ^0.8.0 (StakingRewards.sol#262)
        - ^0.8.0 (StakingRewards.sol#346)
        - ^0.8.0 (StakingRewards.sol#663)
        - ^0.8.0 (StakingRewards.sol#725)
        - ^0.8.0 (StakingRewards.sol#809)
        - ^0.8.0 (StakingRewards.sol#924)
        - ^0.8.1 (StakingRewards.sol#417)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```



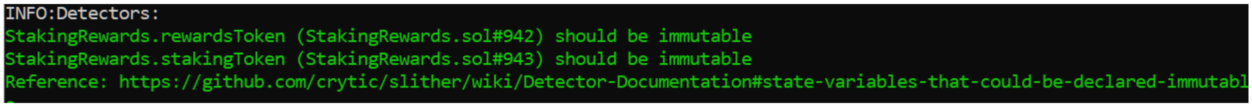
```
INFO:Detectors:
Address.functionCall(address,bytes) (StakingRewards.sol#498-500) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (StakingRewards.sol#527-533) is never used and should be removed
Address.functionDelegateCall(address,bytes) (StakingRewards.sol#583-585) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (StakingRewards.sol#593-600) is never used and should be removed
Address.functionStaticCall(address,bytes) (StakingRewards.sol#558-560) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (StakingRewards.sol#568-575) is never used and should be removed
Address.sendValue(address,uint256) (StakingRewards.sol#473-478) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (StakingRewards.sol#632-642) is never used and should be removed
Context._msgData() (StakingRewards.sol#253-255) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (StakingRewards.sol#850-863) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (StakingRewards.sol#874-885) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (StakingRewards.sol#865-872) is never used and should be removed
SafeERC20.safePermit(IERC20Permit,address,address,uint256,uint256,uint8,bytes32,bytes32) (StakingRewards.sol#887-901) is
 never used and should be removed
SafeMath.div(uint256,uint256,string) (StakingRewards.sol#194-203) is never used and should be removed
SafeMath.mod(uint256,uint256) (StakingRewards.sol#154-156) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (StakingRewards.sol#220-229) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (StakingRewards.sol#171-180) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (StakingRewards.sol#25-31) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (StakingRewards.sol#67-72) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (StakingRewards.sol#79-84) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (StakingRewards.sol#50-60) is never used and should be removed
SafeMath.trySub(uint256,uint256) (StakingRewards.sol#38-43) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```



```
INFO:Detectors:
Pragma version^0.8.0 (StakingRewards.sol#7) allows old versions
Pragma version^0.8.0 (StakingRewards.sol#236) allows old versions
Pragma version^0.8.0 (StakingRewards.sol#262) allows old versions
Pragma version^0.8.0 (StakingRewards.sol#346) allows old versions
Pragma version^0.8.1 (StakingRewards.sol#417) allows old versions
Pragma version^0.8.0 (StakingRewards.sol#663) allows old versions
Pragma version^0.8.0 (StakingRewards.sol#725) allows old versions
Pragma version^0.8.0 (StakingRewards.sol#809) allows old versions
Pragma version^0.8.0 (StakingRewards.sol#924) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

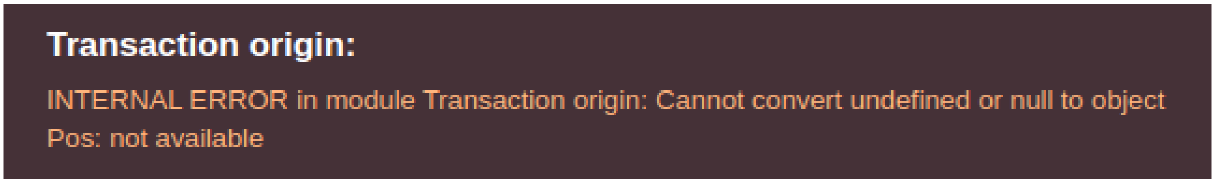


```
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (StakingRewards.sol#473-478):
        - (success) = recipient.call{value: amount}() (StakingRewards.sol#476)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (StakingRewards.sol#541-550):
        - (success,returndata) = target.call{value: value}(data) (StakingRewards.sol#548)
Low level call in Address.functionStaticCall(address,bytes,string) (StakingRewards.sol#568-575):
        - (success,returndata) = target.staticcall(data) (StakingRewards.sol#573)
Low level call in Address.functionDelegateCall(address,bytes,string) (StakingRewards.sol#593-600):
        - (success,returndata) = target.delegatecall(data) (StakingRewards.sol#598)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```



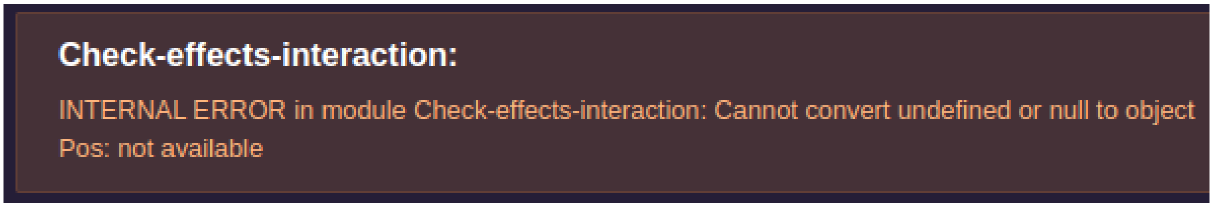
```
INFO:Detectors:
StakingRewards.rewardsToken (StakingRewards.sol#942) should be immutable
StakingRewards.stakingToken (StakingRewards.sol#943) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutabl
```

Solidity Static Analysis



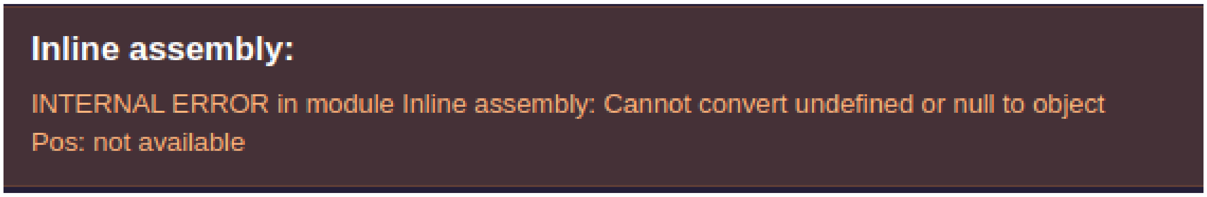
Transaction origin:

INTERNAL ERROR in module Transaction origin: Cannot convert undefined or null to object
Pos: not available



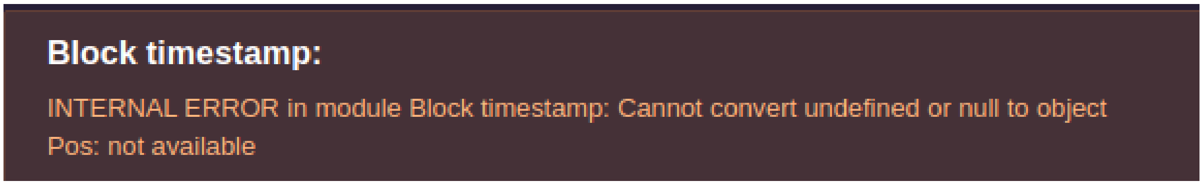
Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: Cannot convert undefined or null to object
Pos: not available



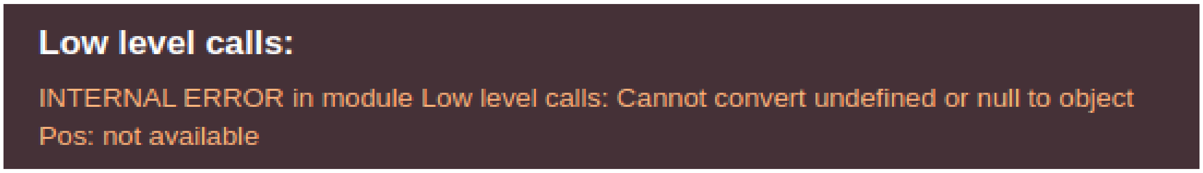
Inline assembly:

INTERNAL ERROR in module Inline assembly: Cannot convert undefined or null to object
Pos: not available



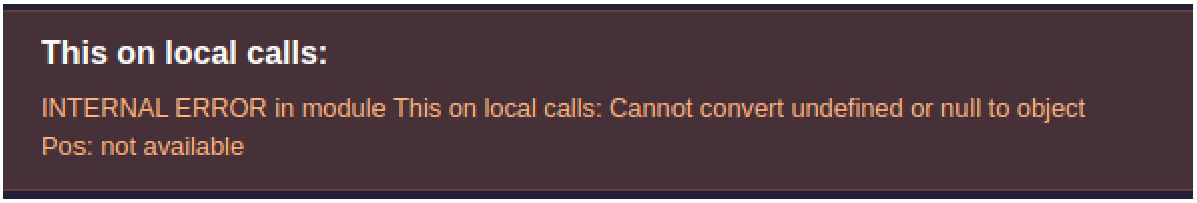
Block timestamp:

INTERNAL ERROR in module Block timestamp: Cannot convert undefined or null to object
Pos: not available



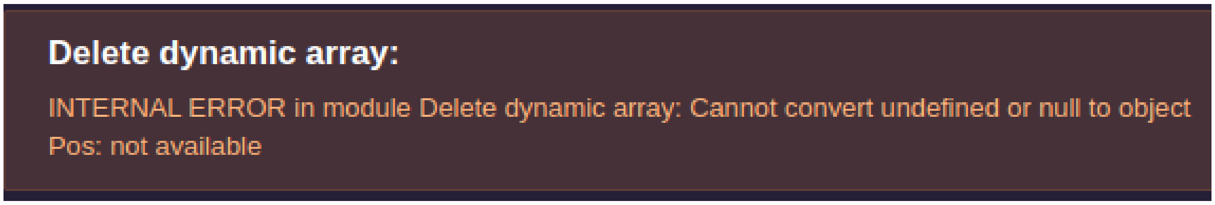
Low level calls:

INTERNAL ERROR in module Low level calls: Cannot convert undefined or null to object
Pos: not available



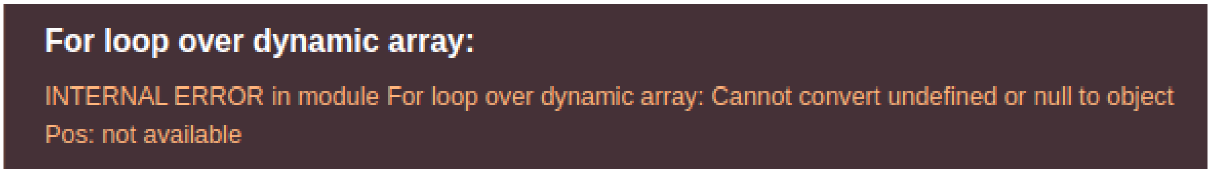
This on local calls:

INTERNAL ERROR in module This on local calls: Cannot convert undefined or null to object
Pos: not available



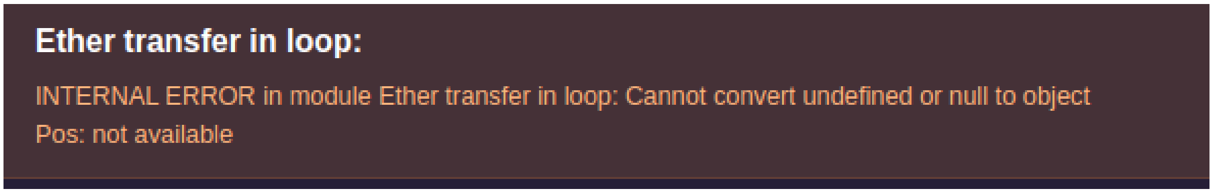
Delete dynamic array:

```
INTERNAL ERROR in module Delete dynamic array: Cannot convert undefined or null to object
Pos: not available
```



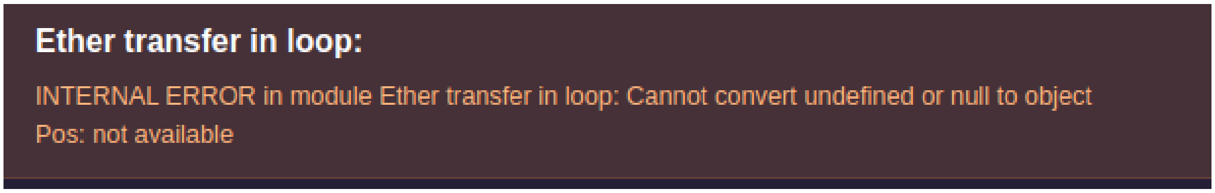
For loop over dynamic array:

```
INTERNAL ERROR in module For loop over dynamic array: Cannot convert undefined or null to object
Pos: not available
```



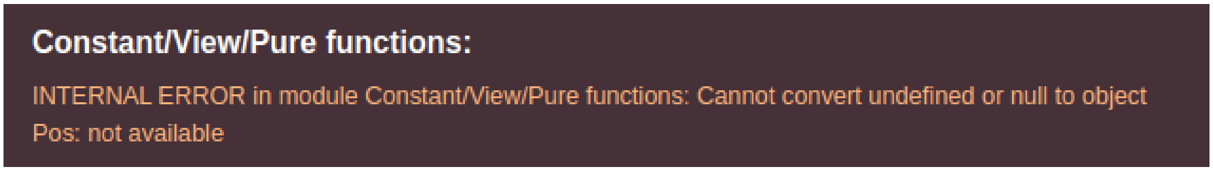
Ether transfer in loop:

```
INTERNAL ERROR in module Ether transfer in loop: Cannot convert undefined or null to object
Pos: not available
```



Ether transfer in loop:

```
INTERNAL ERROR in module Ether transfer in loop: Cannot convert undefined or null to object
Pos: not available
```



Constant/View/Pure functions:

```
INTERNAL ERROR in module Constant/View/Pure functions: Cannot convert undefined or null to object
Pos: not available
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot 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:

No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

Discussion

1) Advised to fix compiler version 0.8.0 instead of ^0.8.0
2) Double check hard coded values before going to production

Conclusion

We were given a contract file and have used all possible tests based on the given object. so it is ready for mainnet deployment. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is “**Secured**”.

Note For Contract Users

The owner wields complete authority over the reward rate, possesses the capability to reclaim assets into their personal custody, and exercises various other pivotal administrative functions with absolute control.

MigrateStaking: This Solidity function, migrateStaking, serves the purpose of allowing the contract owner to migrate staking for a user within the smart contract. It facilitates the addition of a specified amount to a user's staked balance while also updating the total supply accordingly. By verifying that the amount being staked is greater than 0, it ensures the integrity of the staking process. Additionally, it emits a Staked event to provide transparency regarding the staking activity occurring within the contract. The function is designed to be called exclusively by the contract owner, as indicated by the onlyOwner modifier, thereby restricting access to authorized entities for the migration of staking.

NotifyRewardAmount: Only the contract owner can call this function to notify the contract of the amount of rewards to be distributed.

RecoverERC20: Allows the owner to recover ERC20 tokens accidentally sent to the contract, excluding the staking token.

SetRecoveredDuration: This function provides a way for the owner of the contract to adjust the duration of the rewards period, ensuring that the change can only occur when the previous rewards period has completed.

TransferOwnership: The transferOwnership function provides a way for the current owner of the contract to transfer ownership to a new address, as long as that address is not the zero address. This function is crucial for transferring control of the contract to another party,

RenounceOwnership: The renounceOwnership function allows the current owner of the contract to voluntarily renounce their ownership privileges, effectively making the contract ownerless. Once ownership is renounced, it cannot be regained, providing a mechanism for decentralization and preventing any single entity from having control over the contract.

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 analyse 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 scrutinised 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

RD Auditors Disclaimer

The smart contracts given for audit have been analysed 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).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the 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 recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. 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.

RD
AUDITORS

Email: info@rdauditors.com

Website: www.rdauditors.com