



# Project Pi, Contract Code Review and Security Analysis Report

---

Customer: Project Pi  
Prepared on: 4th February 2024  
Platform: Base  
Language: Solidity

[rdauditors.com](https://rdauditors.com)

---

## Table of Contents

<b>Disclaimer</b>	<b>2</b>
<b>Document</b>	<b>3</b>
<b>Introduction</b>	<b>4</b>
<b>Project Scope</b>	<b>5</b>
<b>Executive Summary</b>	<b>6</b>
<b>Code Quality</b>	<b>6</b>
<b>Documentation</b>	<b>8</b>
<b>Use of Dependencies</b>	<b>8</b>
<b>AS-IS Overview</b>	<b>9</b>
<b>Interaction Diagram</b>	<b>14</b>
<b>Inheritance Diagram</b>	<b>14</b>
<b>Code Flow Diagram - Slither Results Log</b>	<b>15</b>
<b>Audit Findings</b>	<b>18</b>
<b>Conclusion</b>	<b>19</b>
<b>Note For Contract Users</b>	<b>19</b>
<b>Our Methodology</b>	<b>23</b>
<b>Disclaimers</b>	<b>25</b>

## 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 Pi
Platform	Base/ Solidity
File	Storage.sol
MD5 hash	640273ac8c751749eb978c0bd31e3cf5
SHA256 hash	e52286a06ec35dd98b6ce268964be37570a0aad5ce0ac0f2be3b2266104d41c4
File	EarlyStaking.sol
MD5 hash	5951470e7d370c4118bb7ab023ed478b
SHA256 hash	e52286a06ec35dd98b6ce268964be37570a0aad5ce0ac0f2be3b2266104d41c4
Date	4/02/2024

## Introduction

RD Auditors (Consultant) were contracted by Project Pi (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 29th January - 4th February 2024.

This contract consists of two 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**.

You are Here

 **Insecure**     **Poorly Secured**     **Secure**     **Well-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

## 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 Project Pi team has 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 Project Pi code as a solidity file.

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

### Pi.sol

#### File And Function Level Report

Contract: Storage

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	setGuardian	write	Passed	All Passed	No Issue	Passed
2	getGuardian	read	Passed	All Passed	No Issue	Passed
3	ConfirmGuardian	write	Passed	All Passed	No Issue	Passed
	registerNetworkContract	write	Passed	All Passed	No Issue	Passed
4	isNetworkContractRegistered	read	Passed	All Passed	No Issue	Passed
5	getAddress	read	Passed	All Passed	No Issue	Passed
6	getBool	read	Passed	All Passed	No Issue	Passed
7	getBytes	read	Passed	All Passed	No Issue	Passed
8	getBytes32	read	Passed	All Passed	No Issue	Passed
9	getInt	read	Passed	All Passed	No Issue	Passed
10	getString	read	Passed	All Passed	No Issue	Passed
11	getUint	read	Passed	All Passed	No Issue	Passed
12	SetAddress	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed

---

13	SetBool	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
14	SetBytes	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
15	SetBytes32	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
16	SetInt	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
17	SetString	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
18	setUInt	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
19	DeleteAddress	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
20	deleteBool	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
21	deleteBytes	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
22	deleteBytes32	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed

---

23	deleteInt	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
24	deleteString	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
25	deleteUint	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
26	addUint	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed
27	subUint	onlyGuardianOrRegisteredNetworkContract	Passed	All Passed	No Issue	Passed

Contract: EarlyStaking2  
 Inherit: Base, ReentrancyGuard  
 Import: Base.sol, storage.sol, TokenPPy  
 Observation: Passed  
 Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	rewardPerToken	read	Passed	All Passed	No Issue	Passed
2	lastRewardTimeStamp	read	Passed	All Passed	No Issue	Passed
3	rewardEarned	read	Passed	All Passed	No Issue	Passed

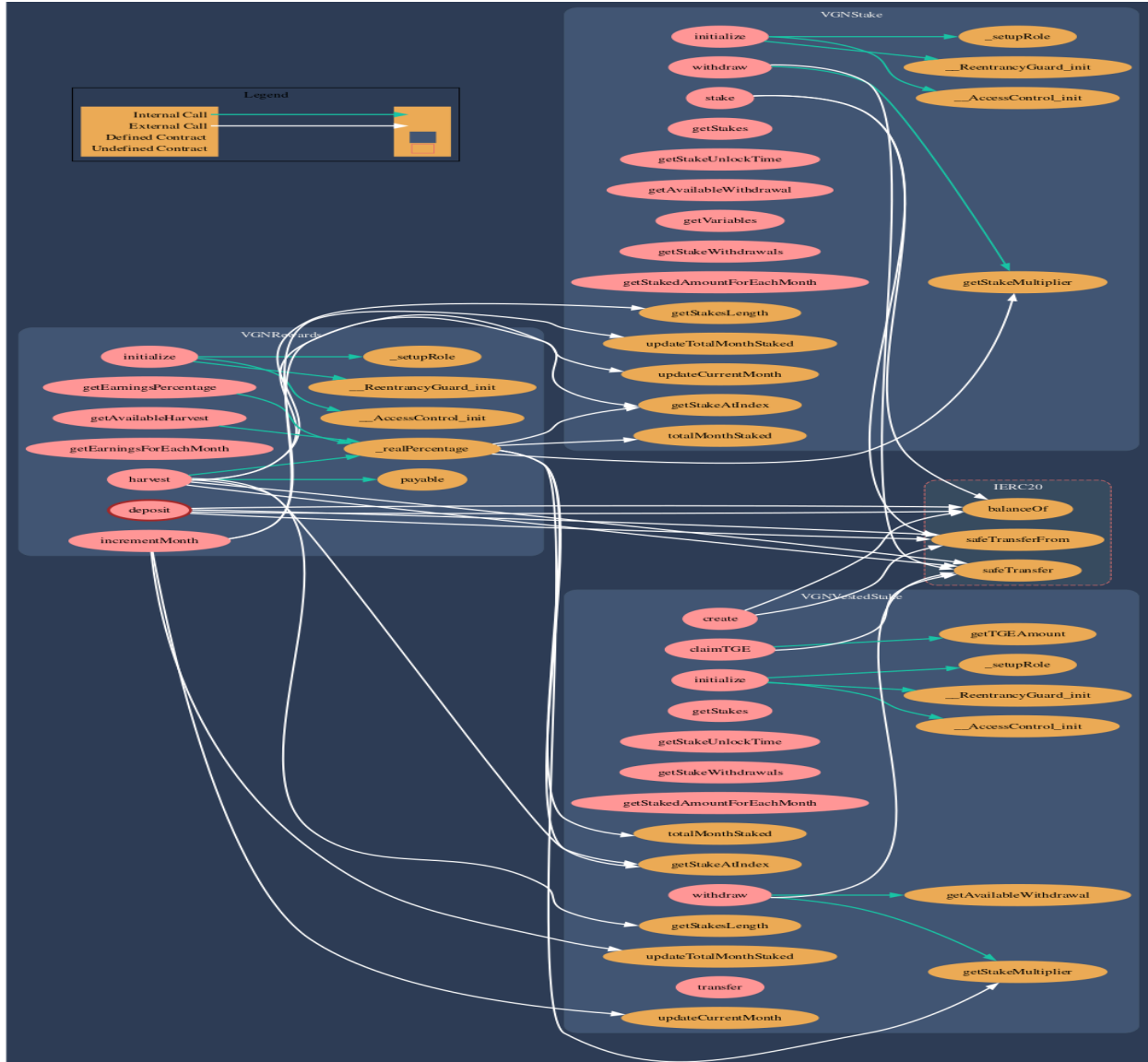
---

4	UpdateRewards	internal	Passed	All Passed	No Issue	Passed
5	Stake	write	Passed	All Passed	No Issue	Passed
6	UnStake	write	Passed	All Passed	No Issue	Passed
7	ClaimRewards	write	Passed	All Passed	No Issue	Passed
8	requiredEarlyStaker	read	Passed	All Passed	No Issue	Passed
9	getStakerCount	read	Passed	All Passed	No Issue	Passed
10	getPLSStake	read	Passed	All Passed	No Issue	Passed
11	getTotalPLSStake	read	Passed	All Passed	No Issue	Passed
12	getIndexOf	read	Passed	All Passed	No Issue	Passed
13	getEarlyStaker	read	Passed	All Passed	No Issue	Passed
14	getEarlyStakers	read	Passed	All Passed	No Issue	Passed

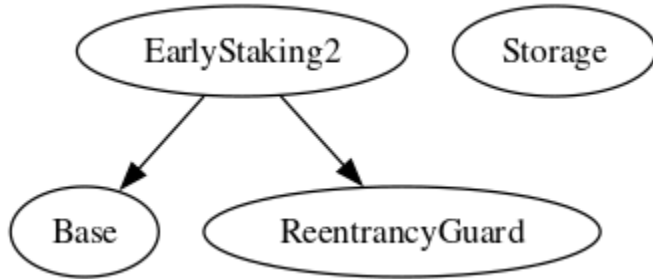
# Code Flow Diagram - Pi

Please follow the below link to see a larger view of the below image:

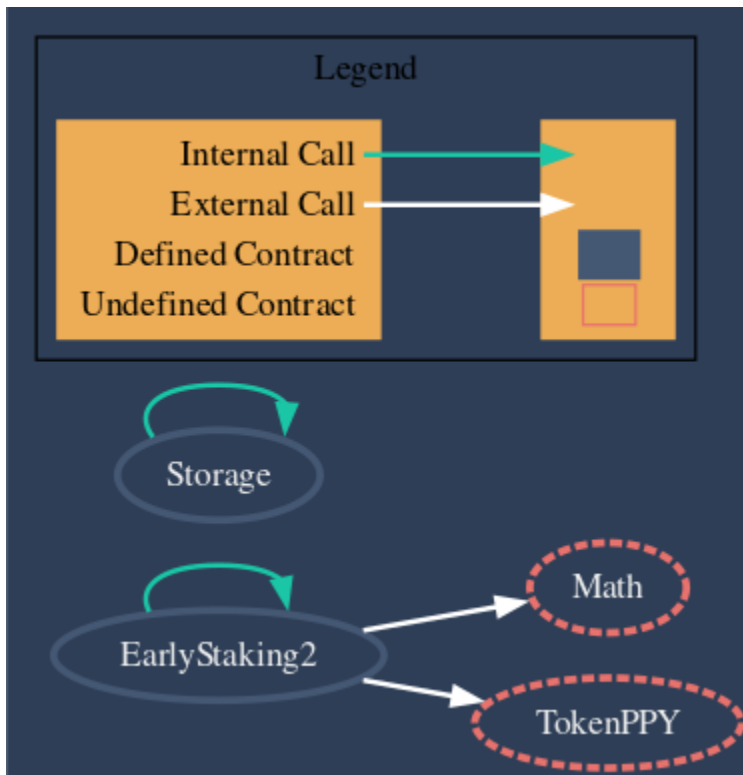
<https://drive.google.com/file/d/17nG-FI6empJqumtmMPebeJkdmRyRduwV/view?usp=sharing>



### Interaction Diagram



### Inheritance Diagram



---

## Code Flow Diagram - Slither Results Log

```
INFO:Detectors:  
Storage.onlyGuardianOrRegisteredNetworkContract() (Storage.sol#28-33) compares to a boolean constant:  
  -booleanStorage[keccak256(bytes)(abi.encodePacked(contract.exists,msg.sender))] == false && msg.sender != guardian (Storage.sol#29)  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality  
INFO:Detectors:  
Pragma version0.8.17 (Storage.sol#2) 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:Slither:Storage.sol analyzed (1 contracts with 93 detectors), 3 result(s) found
```



---

# Solidity Static Analysis

**Gas costs:**

Gas requirement of function Storage.getBytes32 is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 102:1:

**Gas costs:**

Gas requirement of function Storage.getString is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 110:1:

**Gas costs:**

Gas requirement of function Storage.setAddress is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 122:1:

**Gas costs:**

Gas requirement of function Storage.setBytes32 is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 134:1:

**Delete dynamic array:**

The "delete" operation when applied to a dynamically sized array in Solidity generates code to delete each of the elements contained. If the array is large, this operation can surpass the block gas limit and raise an OOG exception. Also nested dynamically sized objects can produce the same results.

[more](#)

Pos: 175:2:

**Similar variable names:**

Storage.getBytes(bytes32) : Variables have very similar names "bytesStorage" and "bytes32Storage". Note: Modifiers are currently not considered by this static analysis.

Pos: 99:9:

**Similar variable names:**

Storage.getBytes32(bytes32) : Variables have very similar names "bytesStorage" and "bytes32Storage". Note: Modifiers are currently not considered by this static analysis.

Pos: 103:9:

**Similar variable names:**

Storage.getUint(bytes32) : Variables have very similar names "intStorage" and "uintStorage". Note: Modifiers are currently not considered by this static analysis.

Pos: 115:9:

---

## 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

Hard coded values must be double checked 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

**SetAddress:** This function allows the guardian or a registered network contract to set an address value in the addressStorage mapping, subject to the access control provided by the onlyGuardianOrRegisteredNetworkContract modifier.

**SetBool:** This function allows the guardian or a registered network contract to set a boolean value in the booleanStorage mapping, subject to the access control provided by the onlyGuardianOrRegisteredNetworkContract modifier.

**SetBytes:** This function allows the guardian or a registered network contract to set a bytes value in the bytesStorage mapping, subject to the access control provided by the onlyGuardianOrRegisteredNetworkContract modifier. This is useful for storing arbitrary binary data in the contract's storage.

**SetBytes32:** This function allows the guardian or a registered network contract to set a bytes32 value in the bytes32 Storage mapping, subject to the access control provided by the onlyGuardianOrRegisteredNetworkContract modifier. This can be useful for storing fixed-size binary data or identifiers in the contract's storage.

**SetInt:** This function allows the guardian or a registered network contract to set an int256 value in the intStorage mapping, subject to the access control provided by the onlyGuardianOrRegisteredNetworkContract modifier.

**SetString:** This function allows the guardian or a registered network contract to set a string value in the stringStorage mapping, subject to the access

control provided by the `onlyGuardianOrRegisteredNetworkContract` modifier. This is useful for storing string data in the contract's storage.

`SetInt`: This function allows the guardian or a registered network contract to set an `int256` value in the `intStorage` mapping, subject to the access control provided by the `onlyGuardianOrRegisteredNetworkContract` modifier. This function is useful for storing integer values in the contract's storage.

`DeleteAddress`: This function allows the guardian or a registered network contract to delete an entry from the `addressStorage` mapping, subject to the access control provided by the `onlyGuardianOrRegisteredNetworkContract` modifier. This can be useful for managing the data stored in the contract and cleaning up unnecessary entries.

`DeleteBool`: This function allows the guardian or a registered network contract to delete an entry from the `booleanStorage` mapping, subject to the access control provided by the `onlyGuardianOrRegisteredNetworkContract` modifier. This can be useful for managing the data stored in the contract and cleaning up unnecessary entries.

`DeleteBytes`: This function allows the guardian or a registered network contract to delete an entry from the `addressStorage` mapping, subject to the access control provided by the `onlyGuardianOrRegisteredNetworkContract` modifier. This can be useful for managing the data stored in the contract and cleaning up unnecessary entries.

`DeleteBytes32`: This function allows the guardian or a registered network contract to delete an entry from the `bytes32` Storage mapping, subject to the access control provided by the `onlyGuardianOrRegisteredNetworkContract`

---

modifier. This can be useful for managing the data stored in the contract and cleaning up unnecessary entries.

**AddUint:** This function allows the guardian or a registered network contract to add a specified amount to the uint256 value stored in the uintStorage mapping at a given key, subject to the access control provided by the `onlyGuardianOrRegisteredNetworkContract` modifier. This is useful for updating and modifying stored integer values in the contract's storage.

**SubUint:** This function allows the guardian or a registered network contract to subtract a specified amount from the uint256 value stored in the uintStorage mapping at a given key, subject to the access control provided by the `onlyGuardianOrRegisteredNetworkContract` modifier. This is useful for updating and modifying stored integer values in the contract's storage.

## 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.



**Email: [info@rdauditors.com](mailto:info@rdauditors.com)**

**Website: [www.rdauditors.com](http://www.rdauditors.com)**

