RD AUDITORS

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: BNFY Finance
Prepared on: 3/04/2021
Platform: Binance Smart Chain
Language: Solidity

info@rdauditors.com

TABLE OF CONTENTS

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 DECISION OF CUSTOMER.

info@rdauditors.com

Document

Name	Smart Contract Code Review and Security Analysis Report for BNFY Finance
Platform	BSC/Solidity
File 1	BNFYStakingNFTV.sol
MD5 hash	7538A9082C9F9C257A4423962BF0072E
SHA256 hash	ABB65D2EA3CBBBEFDFFF534D59AF27FB74F248022A93A20151754AD505466C66
File 2	BNFYStakingv2.sol
MD5 hash	7538A9082C9F9C257A4423962BF0072E
SHA256 hash	ABB65D2EA3CBBBEFDFFF534D59AF27FB74F248022A93A20151754AD505466C66
File 3	LPStakingNFTV2.
MD5 hash	924754FEE4D0256BE04601E65E2ACF75
SHA256 hash	0AC35F7416F95FC9DFC9B4FB52AB7888E8E5BB35E5EF13F359F9AD8FAD1539B5
File 4	LPStakingV2.sol
MD5 hash	AFB6081D9561008D15A142215A0B53B3
SHA256 hash	8A30730CB965BBE26086618CECB7F313DDB280EF74FF265E7719FF5462EE64D3
File 5	RewardPool.sol
MD5 hash	AB9A2A6C9DAD1811CB7ADE08E4D30DF3
SHA256 hash	B197DFAEF7C4752E9C2528393D846181FA8611649ED79189E0B368F98A3621EE
Date	**3/04/2021**

Introduction

RD Auditors (Consultant) was contracted by BNFY Finance (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between March 23 , 2021 – April 3, 2021.

This contract consists of five files.

Project Scope

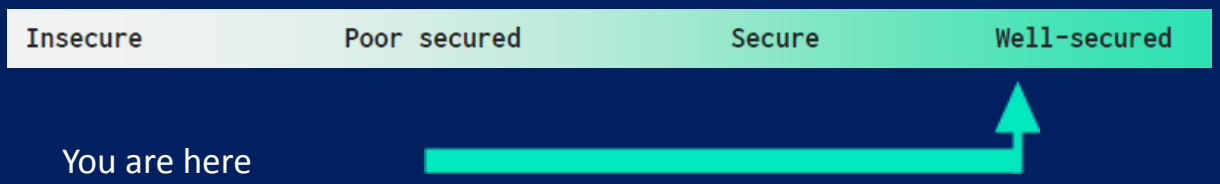
The scope of the project is a smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but 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 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, 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 0 critical, 0 high, some medium, 0 low and 0 very low level issues.

UPDATE (9/04/2021): After modification of code we found 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

BNFY consists of multiple smart contract files. BNFY consists of five smart contract files. This multiple file smart contract also contains safeMath, address library and SafeERC20 etc from the popular open source.

The libraries in the BNFY 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 BNFY.

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

Documentation

We were given 5000 lines of code in the form of a zip file.

The hash of that file is mentioned in the table. As mentioned, It is well commented 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 are written well and systematically.

AS-IS Overview

BNFY Overview

This contract provides financial services to its users with ERC721 standard included for the purpose like deposit reward and withdrawal etc.

File And Function Level Report

File: Ownable.sol

Contract: Ownable
Import: Context
Observation: All Passed including security check
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	transferOwnership	write	Passed	All Passed	No Issue	Passed
2	addplatformAddress	write	Passed	All Passed	No Issue	Passed
3	removeplatformAddress	write	Passed	All Passed	No Issue	Passed

Contract: BNFYStaking
Inherit: ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getRewardPerBlock	read	Passed	All Passed	No Issue	Passed
2	SetDailyReward	write	Passed	All Passed	No Issue	Passed
3	getNFTBalance	read	Passed	All Passed	No Issue	Passed
4	checkIfNFTInCirculation	read	Passed	All Passed	No Issue	Passed
5	PendingRewards	read	Passed	All Passed	No Issue	Passed
6	getTotalRewards	read	Passed	All Passed	No Issue	Passed
7	getTotalBalance	read	Passed	All Passed	No Issue	Passed
8	UpdatePool	read	Passed	All Passed	No Issue	Passed

9	StakeBNFY	read	Passed	All Passed	No Issue	Passed
10	addStakeholder	read	Passed	All Passed	No Issue	Passed
11	addStakeholderExternal	write	Passed	All Passed	No Issue	Passed
12	ClaimRewards	write	Passed	All Passed	No Issue	Passed
13	CompoundRewards	write	Passed	All Passed	No Issue	Passed
14	ClaimAllRewards	write	Passed	All Passed	No Issue	Passed
15	CompoundAllRewards	write	Passed	All Passed	No Issue	Passed
16	UnStakeBNFY	write	Passed	All Passed	No Issue	Passed
17	UnStakeAll	write	Passed	All Passed	No Issue	Passed
18	IncrementNFTValue	write	Passed	All Passed	No Issue	Passed
19	decrementNFTValue	write	Passed	All Passed	No Issue	Passed

File: BNFYStakingV2.sol

Contract: Ownable

inherit: context

Observation: All Passed including security check

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	transferOwnership	write	Passed	All Passed	No Issue	Passed
2	addplatformAddress	write	Passed	All Passed	No Issue	Passed
3	removeplateformAddress	write	Passed	All Passed	No Issue	Passed

Contract: BNFYStakingV2

Inherit: ownable

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getRewardPerBlock	read	Passed	All Passed	No Issue	Passed
2	SetDailyReward	write	Passed	All Passed	No Issue	Passed

3	getNFTBalance	read	Passed	All Passed	No Issue	Passed
4	checkIfNFTInCirculation	read	Passed	All Passed	No Issue	Passed
5	PendingRewards	read	Passed	All Passed	No Issue	Passed
6	getTotalRewards	read	Passed	All Passed	No Issue	Passed
7	getTotalBalance	read	Passed	All Passed	No Issue	Passed
8	UpdatePool	read	Passed	All Passed	No Issue	Passed
9	StakeBNFY	read	Passed	All Passed	No Issue	Passed
10	addStakeholder	read	Passed	All Passed	No Issue	Passed
11	addStakeholderExternal	write	Passed	All Passed	No Issue	Passed
12	ClaimRewards	write	Passed	All Passed	No Issue	Passed
13	CompoundRewards	write	Passed	All Passed	No Issue	Passed
14	ClaimAllRewards	write	Passed	All Passed	No Issue	Passed
15	CompoundAllRewards	write	Passed	All Passed	No Issue	Passed
16	UnStakeBNFY	write	Passed	All Passed	No Issue	Passed
17	UnStakeAll	write	Passed	All Passed	No Issue	Passed
18	IncrementNFTValue	write	Passed	All Passed	No Issue	Passed
19	decrementNFTValue	write	Passed	All Passed	No Issue	Passed

File:LPStakingNFTV.sol

Contract: ERC165
Import: IERC165
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	transferOwnership	write	Passed	All Passed	No Issue	Passed
2	addplatformAddress	write	Passed	All Passed	No Issue	Passed
3	removeplateformAddress	write	Passed	All Passed	No Issue	Passed

Contract: ERC721
Inherit: Context, ERC165, IERC721, IERC721Metadata, IRC721Enumerable
Observation:All Passed including security check
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	balanceOf	read	Passed	All Passed	No Issue	Passed
2	OwnerOf	read	Passed	All Passed	No Issue	Passed
3	name	read	Passed	All Passed	No Issue	Passed
4	symbol	read	Passed	All Passed	No Issue	Passed
5	tokenURI	read	Passed	All Passed	No Issue	Passed
6	baseURI	read	Passed	All Passed	No Issue	Passed
7	tokenOfOwnerByIndex	read	Passed	All Passed	No Issue	passed
8	TotalSupply	write	Passed	All Passed	No Issue	Passed
9	tokenByIndex	write	Passed	All Passed	No Issue	Passed
10	approve	write	Passed	All Passed	No Issue	Passed
11	getapproved	write	Passed	All Passed	No Issue	Passed
12	SetApprovelForAll	read	Passed	All Passed	No Issue	Passed
13	isApprovedForAll	read	Passed	All Passed	No Issue	Passed
14	transferFrom	read	Passed	All Passed	No Issue	Passed
15	SafeTransferFrom	read	Passed	All Passed	No Issue	passed
16	_safetransfer	write	Passed	All Passed	No Issue	passed
17	_exists	read	Passed	All Passed	No Issue	Passed
18	_isApprovedOrOwner	read	Passed	All Passed	No Issue	Passed
19	_SafeMint	write	Passed	All Passed	No Issue	Passed
20	_mint	write	Passed	All Passed	No Issue	Passed
21	_burn	write	Passed	All Passed	No Issue	Passed
22	_transfer	write	Passed	All Passed	No Issue	Passed
23	_setTokenURI	write	Passed	All Passed	No Issue	passed
24	_setBaseURI	write	Passed	All Passed	No Issue	Passed
25	_checkERc721Received	write	Passed	All Passed	No Issue	Passed
26	_approve	write	Passed	All Passed	No Issue	Passed
27	_beforeTokenTransfer	write	Passed	All Passed	No Issue	Passed

Contract: Ownable

inherit: context

Observation: All Passed including security check

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	transferOwnership	write	Passed	All Passed	No Issue	Passed

2	addplatformAddress	write	Passed	All Passed	No Issue	Passed
3	removeplateformAddress	write	Passed	All Passed	No Issue	Passed

Contract: LPStakingNFTV2

inherit: Ownable, ERC721

Observation: All Passed including security check

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	mint	write	Passed	All Passed	No Issue	Passed
2	revertNftTokenId	write	Passed	All Passed	No Issue	Passed
3	NFtTokenId	read	Passed	All Passed	No Issue	Passed
4	burn	write	Passed	All Passed	No Issue	Passed

File:LPStakingV2.sol

Contract: Ownable

inherit: context

Observation: All Passed including security check

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	transferOwnership	write	Passed	All Passed	No Issue	Passed
2	addplatformAddress	write	Passed	All Passed	No Issue	Passed
3	removeplateformAddress	write	Passed	All Passed	No Issue	Passed

Contract: LPStakingV2

Inherit: LPStakingV2

Observation:All Passed including security check

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getRewardPerBlock	read	Passed	All Passed	No Issue	Passed
2	setDailyReward	write	Passed	All Passed	No Issue	Passed
3	getNFTBalance	read	Passed	All Passed	No Issue	Passed
4	checkIfNFTInCirculation	read	Passed	All Passed	No Issue	Passed
5	PendingRewards	read	Passed	All Passed	No Issue	Passed
6	getTotalRewards	read	Passed	All Passed	No Issue	Passed
7	getTotalBalance	read	Passed	All Passed	No Issue	passed
8	updatePool	write	Passed	All Passed	No Issue	Passed
9	StakeBNFY	write	Passed	All Passed	No Issue	Passed
10	addStakeholder	write	Passed	All Passed	No Issue	Passed
11	addStakeholderExternal	write	Passed	All Passed	No Issue	Passed
12	ClaimRewards	write	Passed	All Passed	No Issue	passed
13	CompoundRewards	write	Passed	All Passed	No Issue	Passed
14	ClaimAllRewards	write	Passed	All Passed	No Issue	Passed
15	CompoundAllRewards	write	Passed	All Passed	No Issue	passed
16	UnstakeBNFY	write	Passed	All Passed	No Issue	passed
17	UnstakeAll	write	Passed	All Passed	No Issue	Passed
18	IncrementNFTvalue	write	Passed	All Passed	No Issue	Passed
19	DecrementNFTValue	write	Passed	All Passed	No Issue	Passed

File:RewardPool.sol

Contract: Ownable

inherit: context

Observation: All Passed including security check

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	transferOwnership	write	Passed	All Passed	No Issue	Passed
2	addplatformAddress	write	Passed	All Passed	No Issue	Passed
3	removeplateformAddress	write	Passed	All Passed	No Issue	Passed

Contract: RewardPool

Inherit: Ownable

Observation:All Passed including security check

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	allowTransferToStaking	write	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lossed 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 functions
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

High

No high severity vulnerabilities were found.

Medium

File: BNFYStakingNFTV.sol

1. In mint () address _minter needs to be checked as not equals to address (0).

2. Check TokenId exists in burn().

```
1512    function _burn(uint256 tokenId) internal virtual {
1513        address owner = ownerOf(tokenId);
1514
1515        _beforeTokenTransfer(owner, address(0), tokenId);
1516
1517        // Clear approvals
1518        _approve(address(0), tokenId);
1519
1520        // Clear metadata (if any)
1521        if (bytes(_tokenURIs[tokenId]).length != 0) {
1522            delete _tokenURIs[tokenId];
1523        }
1524
1525        _holderTokens[owner].remove(tokenId);
1526
1527        _tokenOwners.remove(tokenId);
1528
1529        emit Transfer(owner, address(0), tokenId);
1530    }
```

3. _setTokenURI() needs to be public and only owner callable instead of inernal.

```
1567    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal virtual {
1568        require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
1569        _tokenURIs[tokenId] = _tokenURI;
1570    }
```

4. _setBaseURI() needs to be public and only owner callable instead of internal.

```
1577    function _setBaseURI(string memory baseURI_) internal virtual {
1578        _baseURI = baseURI_;
1579    }
```

5. Can use safeguard to all token manipulation functions to pause and unpause contract activity.

UPDATE (9/04/2021): After our suggestion, shortcomings were removed and now there is no vulnerability.

File:BNFYStakingV2.sol

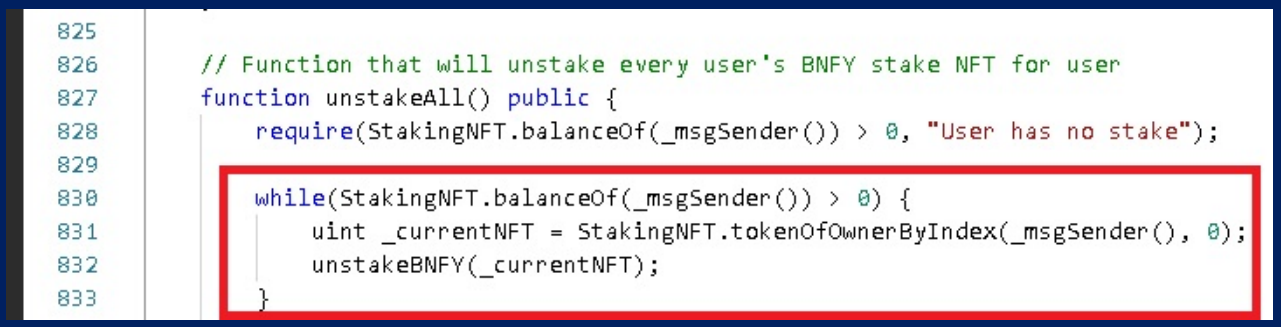
1. Static timestamp declaration in constructor.

```
592    // Constructor will set the address of BNFY token and addre
593    constructor(address _BNFYToken, address _StakingNFT, addres
594        BNFYToken = IERC20(_BNFYToken);
595        StakingNFT = IBNFYStakingNFT(_StakingNFT);
596        staking = _staking;
597        rewardPool = _rewardPool;
598
599        // 9:30 EST December 27th
600        lastRewardBlock = 11536400;
601
602        setDailyReward(_dailyReward);
603        accBNfyPerShare;
604    }
605
```

2. With each stake users do updatepool() getting called. It can be optimized using some condition like after some block it will get updated.

3. Excessive call of updatepool().

4. Need to set some limit in loop in unstakeAll() and compoundAllRewards() and claimAllRewards() otherwise when the user has some huge number of stake with revert this function.



```
825
826     // Function that will unstake every user's BNFY stake NFT for user
827     function unstakeAll() public {
828         require(StakingNFT.balanceOf(_msgSender()) > 0, "User has no stake");
829
830         while(StakingNFT.balanceOf(_msgSender()) > 0) {
831             uint _currentNFT = StakingNFT.tokenOfOwnerByIndex(_msgSender(), 0);
832             unstakeBNFY(_currentNFT);
833         }
```

5.Can use safeguard to all token manipulate functions to pause and unpause contract activity.

UPDATE (9/04/2021): After modification of code we did not get any issues.

File:LPStakingNFTV.sol

1. _setTokenURI() needs to be public and only owner callable instead of internal.
2. _setBaseURI() needs to be public and only owner callable instead of internal.
3. In mint() address _minter needs to be checked as not equals to address(0).
4. Check TokenId exists in burn().
5.Can use safeguard to all token manipulation functions to pause and unpause contract activity.

UPDATE (9/04/2021): After modification of code there is no shortage here.

File:LPStakingV2.sol

1. Remove Static timestamp declaration.
2. Excess use of updatpool().
3. Need to set some limit in loop in unstakeAll() and claimAllRewards() otherwise when the user having some huge number of stake will revert this function.
4. There is no function to turn off the emergency withdrawal. turnEmergencyWithdrawOn() only turns on the emergency withdrawal.
5. Can use safeguard to all token manipulation functions to pause and unpause contract activity.

UPDATE (9/04/2021): After modification of code there is no shortage here.

File:RewardPool.sol

1. Check address and amount not blank in allowTransferToStaking().

```
552     function allowTransferToStaking(address _stakingAddress, uint256 _amount) public onlyOwner() {
553         BNFYToken.approve(_stakingAddress, _amount);
554     }
```

UPDATE (9/04/2021): After modification of code there is no shortage here.

Low

No Low severity vulnerabilities were found.

Very Low

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically but comments were missing. We found some medium issues which has resolved and is now ready to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such extensive smart contract 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.

The security state of the reviewed contract is “well 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

RD Auditors Disclaimer

The smart contracts given for audit have been analyzed 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, so 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 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 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.

RD
AUDITORS

info@rdauditors.com