RD
AUDITORS

StrongNode Smart Contract, Code Review and Security Analysis Report

Customer: StrongNode
Prepared on: 17th October 2021
Platform: Polygon
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 StrongNode
Platform	Polygon / Solidity
File 1	BEP20.sol
MD5 hash	B68ACC1F2595E8F0D3EFC396A92EC49B
SHA256 hash	253F71764BBE8184231BFFF9E7C8A777044D0C5661077554F0C435F38902C61B
File 2	Factory.sol
MD5 hash	409E1CFD0584DEF18F3761A1BD5AC04F
SHA256 hash	58B7CD930B9C1E735881615208BCB6AD826F543EA209A6E63E6BEA5D541602B6
File 3	StrongNodeStake.sol
MD5 hash	69565E0A231D55EDB743760E43DAF3E2

SHA256 hash	627759EA2C77BEADA8E4B44E4B377A02726CB6CF8F7B17D7FD43D14AD256C3C0
File 4	StrongNodeToken.sol
MD5 hash	05EDB1A12F8053A6D8D1861DFB38F2BD
SHA256 hash	DC3F12E99758BAC44D03ED16BDD89E646DE11234BBD311F6FA91DF1C7532908D
File 5	TestToken.sol
MD5 hash	EEFDC90DE9E1BB190C47C7B67768467C
SHA256 hash	16416E0AAE36B2229CB761AF06CCF4285BD9B596FA71B04B1B92366621C83ED3
File 6	XStrongNode.sol
MD5 hash	3C52277A6EFD3E8FBD6A1B04BE5FF28A
SHA256 hash	29DE65805C8682D9C29A83BB3DD35E4CF58F2B70D727937276D3A936D17976F9
File 7	StrongNodeEdge.sol

MD5 hash	3B1C62447931C7B50F430C75DB14A2A9
SHA256 hash	D47958233AFFF74A6AB2C6685411A6D3 5ED6F1397F729F26BE0B403A2FF0164E
Date	17/10/2021

Introduction

RD Auditors (Consultant) were contracted by StrongNode (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between 11 - 17 Oct 2021.

This contract consists of seven files and other supporting 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 **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 the following;

Total Issues	0
Critical	0
High	0
Medium	0
Low	0
Very Low	4

Code Quality

Please find a link that, within this report, safeMath, IBEP20, ownable, SafeBEP20 and context taken from the popular open source.

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 StrongNode 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 well 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

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

AS-IS Overview

StrongNode

File And Function Level Report

File:	BEP20
Contract:	BEP20
Import:	IBEP20, Ownable, SafeMath, Address
Inherit:	Context, IBEP20, Ownable
Observation:	Passed
Test Report:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getOwner	read	Passed	All Passed	No Issue	Passed
2	name	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	symbol	read	Passed	All Passed	No Issue	Passed
5	totalSupply	read	Passed	All Passed	No Issue	Passed
6	balanceOf	read	Passed	All Passed	No Issue	Passed
7	transfer	write	Passed	All Passed	No Issue	Passed
8	allowance	read	Passed	All Passed	No Issue	Passed
9	approve	write	Passed	All Passed	No Issue	Passed
10	transferFrom	write	Passed	All Passed	No Issue	Passed
11	increaseAllowance	write	Passed	All Passed	No Issue	Passed

12	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
13	_transfer	Internal	Passed	All Passed	No Issue	Passed
14	_mint	Internal	Passed	All Passed	No Issue	Passed
15	_burn	Internal	Passed	All Passed	No Issue	Passed
16	_approve	Internal	Passed	All Passed	No Issue	Passed
17	_burnFrom	Internal	Passed	All Passed	No Issue	Passed

File: Factory
Contract: Factory
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	createPair	Internal	Passed	All Passed	No Issue	Passed
2	allPair	Internal	Passed	All Passed	No Issue	Passed
3	allPairLength	Internal	Passed	All Passed	No Issue	Passed

File: StrongNodeStake
Contract: StrongNodeStake
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	hasClaimed	read	Passed	All Passed	No Issue	Passed
2	openStaking	OnlyOwner	Passed	All Passed	No Issue	Passed
3	closeStaking	OnlyOwner	Passed	All Passed	No Issue	Passed
4	UnlockRewards	OnlyOwner	Passed	All Passed	No Issue	Passed
5	lockRewards	OnlyOwner	Passed	All Passed	No Issue	Passed
6	CalculateRewards	OnlyOwner	Passed	All Passed	No Issue	Passed
7	SetMinimumStakeAmount	OnlyOwner	Passed	All Passed	No Issue	Passed
8	SetStrongNodeTokenAddress	OnlyOwner	Passed	All Passed	No Issue	Passed
9	lockTokens	OnlyOwner	Passed	All Passed	No Issue	Passed
10	UnlockTokens	OnlyOwner	Passed	All Passed	No Issue	Passed
11	enterStaking	write	Passed	All Passed	No Issue	Passed
12	ClaimRewards	write	Passed	All Passed	No Issue	Passed

13	LeaveStaking	write	Passed	All Passed	No Issue	Passed
14	emergencyWithdrawCake	write	Passed	All Passed	No Issue	Passed
15	emergencyWithdrawStrong	write	Passed	All Passed	No Issue	Passed
16	ChangeXStrongNodeOwnership	write	Passed	All Passed	No Issue	Passed

File: StrongNodeToken
Contract: StrongNodeToken
Inherit: Context, Ownable, IBEP20
Import: Context, Ownable, Address, SafeMath, IBEP20, IPancakePair
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	applyInitialSellTax	write	Passed	All Passed	No Issue	Passed
2	removeInitialSellTax	write	Passed	All Passed	No Issue	Passed
3	SetSafeAddress	write	Passed	All Passed	No Issue	Passed
4	SetStrong4GoodAddress	write	Passed	All Passed	No Issue	Passed
5	SetTokenPairAddress	write	Passed	All Passed	No Issue	Passed

6	name	read	Passed	All Passed	No Issue	Passed
7	symbol	read	Passed	All Passed	No Issue	Passed
8	decimals	read	Passed	All Passed	No Issue	Passed
9	totalSupply	read	Passed	All Passed	No Issue	Passed
10	balanceOf	read	Passed	All Passed	No Issue	Passed
11	transfer	read	Passed	All Passed	No Issue	Passed
12	allowance	write	Passed	All Passed	No Issue	Passed
13	approve	read	Passed	All Passed	No Issue	Passed
14	Transfer From	write	Passed	All Passed	No Issue	Passed
15	IncreaseAllowance	write	Passed	All Passed	No Issue	Passed
16	decreaseAllowance	read	Passed	All Passed	No Issue	Passed
17	getOwner	read	Passed	All Passed	No Issue	Passed
18	isExcluded	read	Passed	All Passed	No Issue	Passed
19	totalFees	read	Passed	All Passed	No Issue	Passed
20	distributeToAllHolders	write	Passed	All Passed	No Issue	Passed
21	reserveBalanceFromTokenAmount	read	Passed	All Passed	No Issue	Passed

22	tokenBalanceFromReserveAmount	read	Passed	All Passed	No Issue	Passed
23	excludeAccount	onlyOwner	Infinite loop possibility	Owner must exclude 100 wallets or less	Passed with client consent	Passed with client consent
24	includeAccount	onlyOwner	Infinite loop possibility	Owner must exclude 100 wallets or less	Passed with client consent	Passed with client consent
25	_approve	write	Passed	All Passed	No Issue	Passed
26	applyExternalTransactionTax	write	Passed	All Passed	No Issue	Passed
27	_transfer	write	Passed	All Passed	No Issue	Passed
28	_transferStandard	write	Passed	All Passed	No Issue	Passed
29	_transferToExcluded	write	Passed	All Passed	No Issue	Passed
30	_transferFromExcluded	write	Passed	All Passed	No Issue	Passed
31	_transferBothExcluded	write	Passed	All Passed	No Issue	Passed
32	_applyFees	write	Passed	All Passed	No Issue	Passed
33	_getValues	write	Passed	All Passed	No Issue	Passed
34	_getTValues	read	Passed	All Passed	No Issue	Passed

35	_getRValues	read	Passed	All Passed	No Issue	Passed
36	getRate	read	Passed	All Passed	No Issue	Passed
37	getCurrentSupply	read	Passed	All Passed	No Issue	Passed

File: TestToken
Contract: TestToken
Inherit: BEP20
Import: BEP20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	mint	onlyowner	Passed	All Passed	No Issue	Passed
2	burn	onlyowner	Passed	All Passed	No Issue	Passed

File: XStrongNode
Contract: XStrongNode
Inherit: context, IBEP20, Ownable
Import: BEP20, Ownable, SafeMath, Address
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getOwner	read	Passed	All Passed	No Issue	Passed

2	name	read	Passed	All Passed	No Issue	Passed
3	decimal	read	Passed	All Passed	No Issue	Passed
4	symbol	read	Passed	All Passed	No Issue	Passed
5	totalSupply	read	Passed	All Passed	No Issue	Passed
6	balanceOf	read	Passed	All Passed	No Issue	Passed
7	transfer	write	Passed	All Passed	No Issue	Passed
8	allowance	read	Passed	All Passed	No Issue	Passed
9	approve	read	Passed	All Passed	No Issue	Passed
10	transferFrom	read	Passed	All Passed	No Issue	Passed
11	increaseAllowance	read	Passed	All Passed	No Issue	Passed
12	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
13	_transfer	Internal	Passed	All Passed	No Issue	Passed
14	_mint	Internal	Passed	All Passed	No Issue	Passed
15	_burn	Internal	Passed	All Passed	No Issue	Passed
16	-approve	Internal	Passed	All Passed	No Issue	Passed
17	burnFrom	Internal	Passed	All Passed	No Issue	Passed
18	mint	Write	Passed	All Passed	No Issue	Passed
19	burn	write	Passed	All Passed	No Issue	Passed

File: StrongNodeedge

Contract: StrongNodeedge

Inherit: context, IMRC20, Ownable

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getOwner	read	Passed	All Passed	No Issue	Passed
2	decimal	read	Passed	All Passed	No Issue	Passed
3	symbol	read	Passed	All Passed	No Issue	Passed
4	name	read	Passed	All Passed	No Issue	Passed
5	totalSupply	read	Passed	All Passed	No Issue	Passed
6	BalanceOf	read	Passed	All Passed	No Issue	Passed
7	transfer	write	Passed	All Passed	No Issue	Passed
8	allowance	read	Passed	All Passed	No Issue	Passed
9	approve	write	Passed	All Passed	No Issue	Passed
10	transferFrom	write	Passed	All Passed	No Issue	Passed
11	increaseAllowance	write	Passed	All Passed	No Issue	Passed
12	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
13	_transfer	internal	Passed	All Passed	No Issue	Passed

14	_approve	internal	Passed	All Passed	No Issue	Passed

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:

File: StrongNodeToken

1) Hardcoded addresses should be double checked before deploying for production.

```
16    address public presaleWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
17    address public stakingWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
18    address public operationsWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
19    address public teamWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
20    address public treasuryWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
21    address public marketingWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
22    address public advisorWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
23    address public publicsaleWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
```

2) Infinite loop possibility.

If the _excluded array length is increased, then it might hit the gas limit. This is an additional function which would be affected by this is: _getCurrentSupply.

```
423   function includeAccount(
424     address account
425   ) external onlyOwner() {
426     require(_isExcluded[account], "Account is already excluded");
427     for (uint256 i = 0; i < _excluded.length; i++) {
428       if (_excluded[i] == account) {
429         _excluded[i] = _excluded[_excluded.length - 1];
430         _circulatingTokenBalance[account] = 0;
431         _isExcluded[account] = false;
432         _excluded.pop();
433         break;
434       }
435     }
436   }
```

Solution: Please keep this excluded wallet to a minimum, ideally under 100 wallets.

File: StrongNodeEdge

(3) Ownership transfer function.

It is good practice to implement an acceptOwnership style to prevent ownership sent to invalid addresses by human error. Code flow similar to below:

```
function transferOwnership(address _newOwner) public onlyOwner {
    newOwner = _newOwner;
}

//the reason for this flow is to protect owners from sending ownership to unintended address due to human error
function acceptOwnership() public {
    require(msg.sender == newOwner);
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
    newOwner = address(0);
}
```

(4) Hardcoded addresses should be double checked before deploying for production.

```
336     address public presaleWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
337     address public stakingWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
338     address public operationsWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239; // combine
339     address public teamWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
340     address public treasuryWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
341     address public marketingWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;  // combine
342     address public advisorWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
343     address public publicsaleWallet = 0x25849726fFB602Bd654e70B6A3A3801A1Ef11239;
344
```

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, so it is 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 an 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 now “well-secured”

Note For Contract Users

There are several owner only functions. Those can be called by the owner's wallet only. So, if the owner's wallet is compromised, then it carries the risk of the contract becoming vulnerable.

1) Owner can withdraw all the funds from the contract by calling the below functions:

```
178
179  /// @notice Withdraws all the sne from this contract's balance to the owner's address.
180  function emergencyWithdrawStrong() external onlyOwner {
181    uint256 sneBalance = IBEP20(sneTokenAddress).balanceOf(address(this));
182    IBEP20(sneTokenAddress).transfer(owner(), sneBalance);
183  }
```

```
173  /// @notice Withdraws all the cake from this contract's balance to the owner's address.
174  function emergencyWithdrawCake() external onlyOwner {
175    uint256 rewardsBalance = IBEP20(rewardsTokenAddress).balanceOf(address(this));
176    IBEP20(rewardsTokenAddress).transfer(owner(), rewardsBalance);
177  }
```

2) Owner can change or transfer the ownership

```
186    function changeXStrongNodeOwnership() external onlyOwner {
187      xStrongNode.transferOwnership(_msgSender());
188    }
```

3) Mint: Owner can mint unlimited tokens.
4) Burn: Owner can burn tokens.
5) IncludeAccount: Owner can include accounts.
6) ExcludeAccount: Owner can exclude accounts.
7) LockToken: Owner can lock the tokens.
8) UnlockToken: Owner can unlock the tokens.
9) UnlockReward: Owner has the power to unlock the rewards.
10) LockReward: Owner has the power to lock rewards.

Owner has full control over the smart contract. Thus, technical auditing does not guarantee the project's ethical side.
Please do your due diligence before investing. Our audit report is never an investment advice.

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

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

Website: www.rdauditors.com