RD AUDITORS

NINJA STARTER SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Fundi Finance
Prepared on: 17 July 2021
Platform: Binance Smart Chain
Language: Solidity

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.

Document

Name	Smart Contract Code Review and Security Analysis Report of Ninja Starter
Platform	BSC / Solidity
File 1	NinjaStarter.sol
MD5 hash	D045C9A67792A48DF2DBAF6D8ADE2481
SHA256 hash	0456BBCEB8B94135BE6EE158131AD5CC155B7B257D508F80170C185A3C3FE9A8
File 2	BEP20.sol
MD5 hash	8400947E5320A4D887BF13E5C777ED16
SHA256 hash	804C31CEC4ADDAC4931F36BEF88BE72509973D8C94C74E043735CC4BBB639962
Date	1707/2021

Introduction

RD Auditors (Consultant) were contracted by Fundi Finance (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 13 - 17 July 2021.

This contract consists of two files.

Project Scope

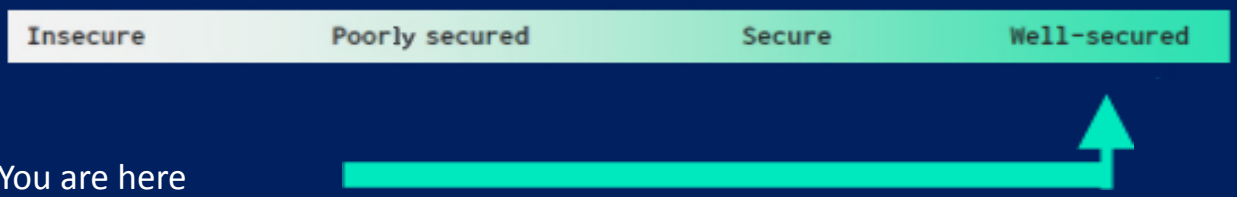
The scope of the project is a smart contract.

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 well-secured.



Update 17.7.21: Issue has been fixed by the dev (file url: https://testnet.bscscan.com/address/0x6f90ec4e3ba08b94bd6cb62bf9cdf63a21e6eb41#code). The contract has now been moved to well-secured from poorly 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 1 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Update 17.7.21: 1 critical issue has been fixed.

Code Quality

Please find a link that, within this report ReentrancyGuard, Ownable, Pausable 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 Fundi Finance team has also conducted unit tests using scripts provided through the same github link which fortify functionality and security of the contract, which also helped us 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 the NinjaStarter contract as a github link:

https://github.com/ninjaswapapp/ninjaswap-core/blob/master/contracts/NinjaStarter.sol

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

Ninja Starter

NinjaStarter is a token sale contract.

File And Function Level Report

File: NinjaStarter

Contract: NinjaStarter
Import: BEP20, SafeBEP20, Ownable, Pausable, ReentrancyGuard
Inherit: ReentrancyGuard, Ownable, Pausable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	buywithBNB	write	Reentrancy	Not passed	Serious Issue Found	Passed after error fixed by dev
2	buywithBUSD	write	Passed	All Passed	No Issue	Passed
3	getEstimatedTokenBuyWithBNB	read	Passed	All Passed	No Issue	Passed
4	_PreValidation	read	Passed	All Passed	No Issue	Passed
5	endOffering	write	Passed	All Passed	No Issue	Passed
6	_getTokenAmount	read	Passed	All Passed	No Issue	Passed
7	_verifyAmount	read	Passed	All Passed	No Issue	Passed
8	setOfferingAmount	read	Passed	All Passed	No Issue	Passed
9	getLatestBNBPrice	read	Passed	All Passed	No Issue	Passed
10	setBuyCap	write	Passed	All Passed	No Issue	Passed
11	setFee	write	Passed	All Passed	No Issue	Passed
12	setPrice	write	Passed	All Passed	No Issue	Passed

13	FinalWithdraw	write	Passed	All Passed	No Issue	Passed
14	recoverStuckTokens	write	Passed	All Passed	No Issue	Passed

File: BEP20.sol

Contract: BEP20
Import: Ownable, Context, IBEP20, SafeMath
Inherit: Context, IBEP20, 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	name	read	Passed	All Passed	No Issue	Passed
3	symbol	read	Passed	All Passed	No Issue	Passed
4	decimals	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	mint	write	Passed	All Passed	No Issue	Passed
14	-transfer	write	Passed	All Passed	No Issue	Passed
15	_mint	write	Passed	All Passed	No Issue	Passed
16	_burn	write	Passed	All Passed	No Issue	Passed
17	_approve	write	Passed	All Passed	No Issue	Passed
18	burnFrom	write	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

1. In line no. 114 reentrancy is possible, .call is used which could possibly forward all available gas.

```
101     function buywithBNB(address _beneficiary)
102         public
103         payable
104         whenNotPaused
105     {
106         uint256 bnbAmount = msg.value;
107         require(bnbAmount > 0, "Please send some more BNB");
108         require(_preValidation(), "offering already finalized");
109         uint256 tokensToBePurchased = _getTokenAmount(bnbAmount);
110         tokensToBePurchased = _verifyAmount(tokensToBePurchased);
111         require(tokensToBePurchased > 0, "You've reached your limit of purchases");
112         uint256 cost = tokensToBePurchased.mul(buyPrice).div(getLatestBNBPrice());
113         if (bnbAmount > cost) {
114             (bool sent, ) = payable(msg.sender).call{value: msg.value - (cost)}("");
115             require(sent);
116             bnbAmount = cost;
117         }
118         // Update total sale participants
119         if (busdDeposits[msg.sender] == 0 && bnbDeposits[_beneficiary] == 0) {
120             totalSaleParticipants = totalSaleParticipants.add(1);
121         }
122         totalCollectedBNB = totalCollectedBNB.add(bnbAmount);
123         offeringToken.safeTransfer(address(msg.sender), tokensToBePurchased);
124         totalSold = totalSold.add(tokensToBePurchased);
125         bnbDeposits[msg.sender] = bnbDeposits[msg.sender].add(bnbAmount);
126         purchases[msg.sender] = purchases[msg.sender].add(tokensToBePurchased);
127         emit purchased(_beneficiary, tokensToBePurchased);
128     }
```

Solution: Recommended to use '.transfer' instead of '.call'.

Update 17.7.21: This error has been fixed and is ready for production.

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. Hardcoded values should be double checked before deploying for production.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically,but we found 1 critical issue so after fixing the issue it 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 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” after a fix was applied.

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