



SECURITY AUDIT

BNBBANKER1

0xFe089BC8E627B5b343969642A979e2A830aEE064

20th May 2022

<https://kalianalytics.com>

Contents

Introduction.....	2
Audited project.....	2
Token Info.....	3
Project Overview	4
Audit Summary	4
Code Quality	5
Documentation.....	5
Conclusion	6
Code Analysis Results.....	6
Findings and recommendations.....	9
Solidity UML Diagram	12
Methodology	13
Manual Code Review	13
Vulnerability Analysis.....	13
Documenting Results.....	13
Suggested Solutions.....	13
DISCLAIMER	14

Introduction

KaliAnalytics is a 3rd party auditing company that works on audits based on client requests. KaliAnalytics was contracted by the Banker.Farm team to perform the security audit of the **BNBBANKER1** Token smart contract code. The audit has been performed using manual analysis as well as using automated tools. This report presents all the findings regarding the audit completed on May 20th, 2022.

The purpose of this audit was to focus on the overall security of the smart contract, reduce residual risk and verify functions and functionality. Furthermore, the KaliAnalytics' audit team focused to identify any security vulnerabilities that may be present and providing feedback based on the findings.

Audited project

Audited project name: BNB Banker / Banker.Farm

Language(s): Solidity

Client contact(s): Banker.Farm Team

Project website: <https://bnb.banker.farm>

Audit team: KaliAnalytics Solidity Audit Team / NEO

Audit completed: 20th May 2022

Audit result: **P A S S E D**

Token Info

Blockchain: Binance Smart Chain (BSC)

Type: BEP20

Contract address: **0xFe089BC8E627B5b343969642A979e2A830aEE064**

Contract Source Code Verified: Yes (Exact Match)

Token name: BnbBanker1

Token ticker: BNBBANKER1

Total supply: 1.000.000

Decimals: 18

Creator address: **0xd1B19fa48539991f79Aa130d7F4dB083bC0CC51F**

Optimization Enabled: Yes with 200 runs

Compiler Version: v0.5.8+commit.23d335f2

Circulating supply as of the date of audit: 10.000

Holders as of the date of the audit: 1

Total transactions as of the date of the audit: 1

Project Overview

BNBBANKER1 smart contract provides a standard BEP20 token. Users can stake **BNB** to receive **BNBBANKER1** tokens, which they can stake to receive **BNBBANKER1** as interest. Users can sell **BNBBANKER1** for BNB. Principal **BNB** deposits cannot be withdrawn. Users can get a referral commission in **BNB**. Dividends are paid from deposits of other users. Liquidity is owned by the smart contract. Users can interact with the smart contract through the project's website. Worth noting, that the team said in correspondence **BNBBANKER1** is not meant for long-term holding, the token is part of the Banker.Farm ecosystem consisting rounds, each having a unique token meant only for short-term speculative holding. Its value will reach zero the moment the TVL reaches the minimum.

Audit Summary

We found 0 critical, 0 high, 0 medium and 2 low, and 0 very low-level issues. The **BNBBANKER1** smart contract is considered **SECURE**, and the audit is **PASSED**.

According to the standard audit assessment, the customer's solidity smart contract is technically secure and there are no flaws in functional correctness. The contract includes owner functions (please refer to the Solidity UML Diagram of the smart contract) reducing the level of decentralization. Thus, the project team must execute those functions as per their business plan.

The KaliAnalytics' Solidity Audit Team used various software tools such as MythX, Slither, and Remix IDE to analyze the smart contract. Furthermore, the team performed a manual audit at a general level.

All issues found during automated analysis were manually reviewed and application vulnerabilities are presented in the Code Analysis Results section

which is presented AS-IS. Identified issues are described at a more detailed level with recommendations in the Findings and Recommendations section.

Code Quality

This audit scope has one smart contract. It contains Libraries, Smart contract inherits, and Interfaces. The scope does not include a business plan, website, team, etc.

The KaliAnalytics' Solidity Audit Team received **BNBBANKER1** Token smart contract code in the form of a BscScan web link.

The **BNBBANKER1** smart contract is compact and professionally written. However, the code is partially not well commented.

Documentation

Comments are extremely helpful in understanding the overall architecture of the protocol. The code is partially not well commented. Therefore, it is or may be difficult to form a general understanding of the logic. However, this does not affect functionality or security.

Another source of information was the project's official website <https://bnb.banker.farm>. Following standard practice, the audit team went through also projects on social media channels, which included **Twitter, Facebook, Telegram, Instagram, and Discord**. The information provided on the website and other channels are standard.

Furthermore, the audit team examined the project's white paper which included all the necessary information about the project including tokenomics, ways to contact the team as well as instructions on how to use the dApp.

Conclusion

The **BNBBANKER1** Smart-Contract was found not to contain technically severe vulnerabilities, no backdoors, no suspicious functions, or no suspicious functionality.

All libraries which were used for calculation and the token in the contract are standard and safe.

The audit team assessed the code with compatible compilers and simulated and manually reviewed for all commonly known and specific vulnerabilities.

Therefore, KaliAnalytics' Solidity Audit team concludes the **BNBBANKER1** Smart-Contract is technically safe for use in the **Binance Smart Chain** main network.

Code Analysis Results

Assertions and Property Checking

Detector	Analyses Type	SWC-ID	STATUS
Solidity assert violation	Symbolic analysis, fuzzing (bytecode)	SWC-110	PASSED
MythX assertion violation (AssertionFailed event)	Symbolic analysis, fuzzing (bytecode)	SWC-110	PASSED

Byte-code Safety

Detector	Analyses Type	SWC-ID	STATUS
Integer overflow in arithmetic operation	Symbolic analysis, fuzzing (bytecode)	SWC-101	PASSED
Integer underflow in arithmetic operation	Symbolic analysis, fuzzing (bytecode)	SWC-101	PASSED
Caller can redirect execution to arbitrary locations	Symbolic analysis, fuzzing (bytecode)	SWC-127	PASSED
Caller can write to arbitrary storage locations	Symbolic analysis, fuzzing (bytecode)	SWC-124	PASSED

Dangerous use of uninitialized storage variables

Solidity code analysis

SWC-109

PASSED

Authorization Controls

Detector	Analyses Type	SWC-ID	STATUS
Any sender can withdraw ETH from the contract account	Symbolic analysis, fuzzing (bytecode)	SWC-105	PASSED
Any sender can trigger SELFDESTRUCT	Symbolic analysis, fuzzing (bytecode)	SWC-106	PASSED
Use of "tx.origin" as a part of authorization control	Solidity code analysis	SWC-115	PASSED

Control Flow

Detector	Analyses Type	SWC-ID	STATUS
Delegatecall to a user-supplied address	Symbolic analysis (bytecode)	SWC-112	PASSED
Call to a user-supplied address	Symbolic analysis (bytecode)	SWC-107	FAILED
Unchecked return value from external call	Solidity code analysis	SWC-104	PASSED
Block timestamp influences a control flow decision	Taint analysis (bytecode)	SWC-116	PASSED
Environment variables influence a control flow decisions	Taint analysis (bytecode)	SWC-120	PASSED
Loop over unbounded data structure	Solidity code analysis	SWC-128	PASSED
Implicit loop over unbounded data structure	Solidity code analysis	SWC-128	PASSED
Usage of "continue" in "do-while"	Solidity code analysis	N/A	PASSED
Multiple calls are executed in the same transaction	Static analysis (bytecode)	SWC-113	PASSED
Persistent state read following external call	Static analysis, fuzzing (bytecode)	SWC-107	PASSED
Persistent state write following external call	Static analysis, fuzzing (bytecode)	SWC-107	PASSED
Account state accessed after call to user-defined address	Symbolic analysis (bytecode)	SWC-107	PASSED

Return value of an external call is not checked	Static analysis (bytecode)	SWC-104	PASSED
Potential weak source of randomness	Solidity code analysis	SWC-120	PASSED
Requirement violation	Fuzzing (bytecode)	SWC-123	FAILED
Call with hardcoded gas amount	Solidity code analysis	SWC-134	PASSED

ERC Standards

Detector	Analyses Type	SWC-ID	STATUS
Incorrect ERC20 implementation	Solidity code analysis	N/A	PASSED

Solidity Coding Best Practices

Detector	Analyses Type	SWC-ID	STATUS
Outdated compiler version	Solidity code analysis	SWC-102	PASSED
No or floating compiler version set	Solidity code analysis	SWC-103	PASSED
Use of right-to-left-override control character	Solidity code analysis	SWC-130	PASSED
Shadowing of built-in symbol	Solidity code analysis	SWC-119	PASSED
Incorrect constructor name	Solidity code analysis	SWC-118	PASSED
State variable shadows another state variable	Solidity code analysis	SWC-119	PASSED
Local variable shadows a state variable	Solidity code analysis	SWC-119	PASSED
Function parameter shadows a state variable	Solidity code analysis	SWC-119	PASSED
Named return value shadows a state variable	Solidity code analysis	SWC-119	PASSED
Unary operation without effect	Solidity code analysis	SWC-129	PASSED
Unary operation directly after assignment	Solidity code analysis	SWC-129	PASSED
Unused state variable	Solidity code analysis	SWC-131	PASSED
Unused local variable	Solidity code analysis	SWC-131	PASSED
Function visibility is not set	Solidity code analysis	SWC-100	PASSED
State variable visibility is not set	Solidity code analysis	SWC-108	PASSED

Use of deprecated functions: <code>callcode()</code> , <code>sha3()</code> , ...	Solidity code analysis	SWC-111	PASSED
Use of deprecated global variables (<code>msg.gas</code> , ...)	Solidity code analysis	SWC-111	PASSED
Use of deprecated keywords (<code>throw</code> , <code>var</code>)	Solidity code analysis	SWC-111	PASSED
Incorrect function state mutability	Solidity code analysis	N/A	PASSED

Findings and recommendations

A call to a user-supplied address is executed. An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place. [SWC-107]

LOW A call to a user-supplied address is executed.

SWC-107 An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file
contracts/BNBBANKER1.sol

Locations

```

253 | require(approve(spender, amount));
254 |
255 | approvedCallFallback(spender, receivedApproval(msg.sender, amount, address(this), extraData);
256 |
257 | return true;

```

Requirement violation. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments). [SWC-123]

LOW

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

SWC-123

Source file

contracts/BNBBANKER1.sol

Locations

```

233 require(approve(spender, amount));
234
235 ApproveAndCallFallback(spender, receiveApproval(msg.sender, amount, address(this), extraData);
236
237 return true;

```

Source file

contracts/BNBBANKER1.sol

Locations

```

228 }
229
230
231 contract Token is ERC20 {
232     mapping(address => bool) private _contracts;
233
234     constructor() public {
235         _name = "BnbBanker1";
236         _symbol = "BNBBANKER1";
237         _decimals = 18;
238         _limitSupply = 1000000000;
239     }
240
241     function approveAndCall(address spender, uint256 amount, bytes memory extraData) public returns (bool) {
242         require(approve(spender, amount));
243
244         ApproveAndCallFallback(spender, receiveApproval(msg.sender, amount, address(this), extraData);
245
246         return true;
247     }
248
249     function transfer(address to, uint256 value) public returns (bool) {
250
251         if (_contracts[to]) {
252             approveAndCall(to, value, new bytes(0));
253         } else {
254             super.transfer(to, value);
255         }
256
257         return true;
258     }
259
260
261
262 contract BnbBanker is Token {

```

Compiler specific version warnings:

The compiled contract might be susceptible to NestedCallataArrayAbiReencodingSizeValidation (very low-severity), ABIDecodeTwoDimensionalArrayMemory (very low-severity), KeccakCaching (medium-severity), EmptyByteArrayCopy (medium-severity), DynamicArrayCleanup (medium-severity), ImplicitConstructorCallvalueCheck (very low-severity), TupleAssignmentMultiStackSlotComponents (very low-severity), MemoryArrayCreationOverflow (low-severity), privateCanBeOverridden (low-severity), YulOptimizerRedundantAssignmentBreakContinue0.5 (low-severity), ABIEncoderV2CalldataStructsWithStaticallySizedAndDynamicallyEncodedMembers (low-severity), SignedArrayStorageCopy (low/medium-severity), ABIEncoderV2StorageArrayWithMultiSlotElement (low-severity), DynamicConstructorArgumentsClippedABIV2 (very low-severity) Solidity Compiler Bugs.

[illegible]

<https://storage.googleapis.com/sol2uml-storage/bsc-0xfe089bc8e627b5b343969642a979e2a830aee064.svg>

Methodology V1.1

The goal of this security audit is to reduce residual risk and improve the security of the smart contract and the quality of its code by identifying possible vulnerabilities and security risks and providing feedback and recommendations based on analysis and findings to help protect users. We use the following methodology in our security audit process.

Manual Code Review

In manual code review, we look for any potential issues with code logic, error handling, cryptographic errors, protocol and header parsing, and random number generators. Although our primary focus is on the in-scope code, we examine dependency code and behavior when relevant.

Vulnerability Analysis

Our audit methods included manual code analysis, automated vulnerability scanners and analyzers, user interface interaction, and penetration testing. We examine the project's website and whitepaper to form a high-level understanding of what functionality the project intends to provide. We may review other audit reports, and similar projects, examine source code dependencies, and generally investigate details other than the implementation.

Documenting Results

We follow our transparent audit process described in this document for analyzing potential security vulnerabilities. Whenever a potential issue is discovered, we immediately create an entry describing the nature of the issue, even though the feasibility and impact of the issue may not yet be verified. Furthermore, we provide recommendations based on findings.

DISCLAIMER V1.1

The audit makes no statements or warranties on the security of the code / the smart contract(s). It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While the KaliAnalytics' Solidity Audit Team has done its best in conducting the analysis, it is important to note that you should not rely on this security audit only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

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 vulnerabilities that can lead to critical issues.

The scope of the audit is strictly only the smart contract code at the specified address. Any concerns about the project must be raised directly to the project team. This document does not constitute an offer to purchase or solicitation to sell, nor is it a recommendation to buy or sell, any token, cryptocurrency, or other product or service. Neither the authors of this document nor any participants in the audit accept any liability for losses or taxes that holders, purchasers, users, or sellers of the token, cryptocurrency, product, or service may incur. You should not construe any information or other material on this document as legal, tax, investment, financial, security, or other advice.

KALIANALYTICS SOLIDITY AUDIT TEAM / NEO

20th May 2022