



# Valid Proof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC**

# Audit

## Security Assessment

December 2021

For



Disclaimer	3
About Us	4
Description	5
Project Engagement	5
Logo	5
Contract Link	5
Vulnerability & Risk Level	6
Methodology	7
Used Code from other Frameworks/Smart Contracts (direct imports)	8
Tested Contract Files	9
Source Lines	10
Risk Level	10
Capabilities	11
Scope of Work	12
Verify Claims	13
Write functions of contract	14
Deployer cannot mint any new tokens	15
Deployer cannot burn or lock user funds	16
Deployer cannot pause the contract	17
Overall checkup (Smart Contract Security)	18
Source Units in Scope	19
Audit Results	20
Audit Comments	21
SWC Attacks	22-25

# Disclaimer

Valid Proof reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. Valid Proof do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

**Valid Proof Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. Valid Proof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.**

Valid Proof Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. Valid Proof’s position is that each company and individual are responsible for their own due diligence and continuous security. Valid Proof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	December 2021	<ul style="list-style-type: none"><li>• Layout project</li><li>• Automated- /Manual-Security</li><li>• Testing</li><li>• Summary</li></ul>

# **Network**

Binance Smart Chain (BEP20)

## **Website**

<https://ninjafloki.io>

## **Telegram**

<https://t.me/ninjaflokitoken>

## **Twitter**

<https://twitter.com/ninjaflokitoken>

## **Instagram**

<https://instagram.com/ninjaflokitoken>

## **GitHub**

<https://github.com/ninjaflokitoken>

## **Reddit**

[https://www.reddit.com/user/ninjafloki\\_token](https://www.reddit.com/user/ninjafloki_token)

## **Medium**

[https://medium.com/@ninjafloki\\_token](https://medium.com/@ninjafloki_token)

## **YouTube**

[https://medium.com/@ninjafloki\\_token](https://medium.com/@ninjafloki_token)

# Description

..... is a platform that integrates NFT gaming and decentralized yield farming applications.

..... ecosystem comprises NFT digital collectibles summoning Metaverse based on Binance Smart Chain (BSC). The game is committed to providing exciting and profitable NFT treats for ..... players. Players here can summon their unique characters for PVP battles, Boss missions, develop into different elements, level up, and ultimately earn profit in the process of the game.

## Project Engagement

During the 16th of December 2021, NinjaFloki Team engaged Volid Proof to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Volid Proof with access to their code repository and whitepaper.

## Logo



## Contract Link

**v1.0**

Contract: <https://bscscan.com/>

address/0x7334186aeA6017A90572a71F43a845820486F766

# Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Information	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

# Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as they were discovered.

## Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:

- i) Review of the specifications, sources, and instructions provided to Valid Proof to make sure we understand the size, scope, and functionality of the smart contract.
- ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Valid Proof describe.

2. Testing and automated analysis that includes the following:

- i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
- ii) Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

# Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:

IBEP20

IBEP20Metadata

Context

Ownable

SafeMath

SafeMathInt

SafeMathUint

# Tested Contract Files

All files and codes of the contract, whose address is below, have been tested.

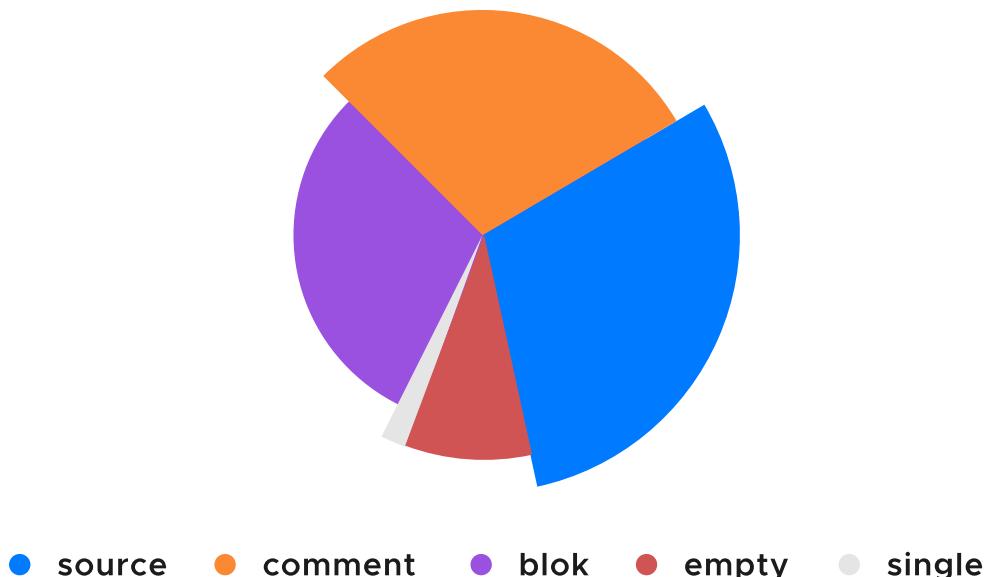
v1.0

File Name	Address
contracts/ninjafloki.sol	0x7334186aeA6017A90572a71F43a845820486F766

# Metrics

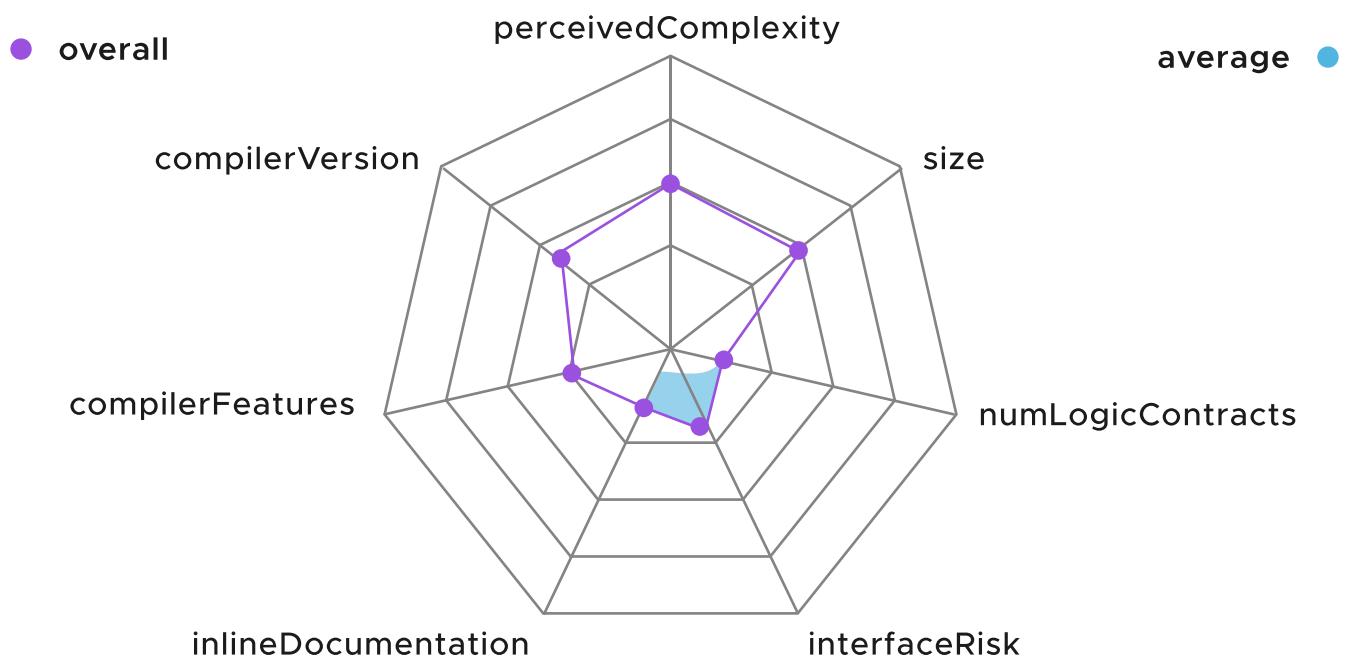
## Source Lines

v1.0



## Risk Level

v1.0



## Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	^0.6.12			**** (Oasm blocks)	

Version	Transfers ETH	Low- Level Calls	Delega teCall	Uses Hash Functions	ECRec over	New/ Create/ Create 2
1.0	yes					

# Scope of Work

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

# Verify Claims

## Correct implementation of Token standard

Tested	Verified
✓	✓

Function	Description	Exist	Tested	Verified
TotalSupply	provides information about the total token supply	✓	✓	✓
BalanceOf	provides account balance of the owner's account	✓	✓	✓
Transfer	executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	returns a set number of tokens from a spender to the owner	✓	✓	✓

# Write functions of contract

1. approve

13. setExcludedFromWhale

2. checkForCoolDown

14. setLiquiditFee

3. claim

15. setMarketingFee

4. decreaseAllowance

16. setMarketingWallet

5. enableCooldown

17. setMaxWalletLimit

6. excludeFromDividends

18. setSellCooldownperiod

7. includeInWhiteList

19. setSellLimitRate

8. increaseAllowance

20. transfer

9. processDividendTracker

21. transferFrom

10. renounceOwnership

22. transferOwnership

11. setAutomatedMarketMakerPair

23. updateClaimWait

12. setBUSDRewardsFee

24. updateDividendTracker

25. updateGasForProcessing

26. updateUniswapV2Router

# Deployer cannot mint any new tokens

Name	Exist	Tested	Verified	File
Deployer cannot mint	✓	✓	✓	Main
Comment	Line: -			

Max / Total Supply: 100.000.000.000

# Deployer cannot burn or lock user funds

Name	Exist	Tested	Verified
Deployer cannot mint	✓	✓	✓
Deployer cannot burn	✓	✓	✓

Comments:

v1.0

- Deployer cannot lock users.

## Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	✓	✓	✓



# Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

## Legend

Attribute	Symbol
Verified / Checked	✓
Partly Verified	🚩
Unverified / Not checked	✗
Not available	—

# Source Units in Scope

v1.0

## Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
nSLOC	normalized source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

# Audit Results

**AUDIT PASSED**



## Security Issues

### Critical Severity Issues

No critical severity issues found.

### High Severity Issues

No medium severity issues found.

### Medium Severity Issues

No medium severity issues found.

### Low Severity Issues

No low severity issues found.

# **Audit Comments**

## **17. December 2021:**

- It is not a contract that will cause problems for the user. This contract, which protects users and consists of certain limits, is reliable in our opinion.

# SWC Attacks

ID	Title	Relationships	Type
<a href="#">SW C-13 6</a>	Unencrypted Private Data On-Chain	<a href="#">CWE-767: Access to Critical Private Variable via Public Method</a>	PASSED
<a href="#">SW C-13 5</a>	Code With No Effects	<a href="#">CWE-1164: Irrelevant Code</a>	PASSED
<a href="#">SW C-13 4</a>	Message call with hardcoded gas amount	<a href="#">CWE-655: Improper Initialization</a>	PASSED
<a href="#">SW C-13 3</a>	Hash Collisions With Multiple Variable Length Arguments	<a href="#">CWE-294: Authentication Bypass by Capture-replay</a>	PASSED
<a href="#">SW C-13 2</a>	Unexpected Ether balance	<a href="#">CWE-667: Improper Locking</a>	PASSED
<a href="#">SW C-13 1</a>	Presence of unused variables	<a href="#">CWE-1164: Irrelevant Code</a>	PASSED
<a href="#">SW C-13 0</a>	Right-To-Left-Override control character (U+202E)	<a href="#">CWE-451: User Interface (UI) Misrepresentation of Critical Information</a>	PASSED
<a href="#">SW C-12 9</a>	Typographical Error	<a href="#">CWE-480: Use of Incorrect Operator</a>	PASSED
<a href="#">SW C-12 8</a>	DoS With Block Gas Limit	<a href="#">CWE-400: Uncontrolled Resource Consumption</a>	PASSED

<a href="#"><u>SW C-12 7</u></a>	Arbitrary Jump with Function Type Variable	<a href="#"><u>CWE-695: Use of Low-Level Functionality</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-12 5</u></a>	Incorrect Inheritance Order	<a href="#"><u>CWE-696: Incorrect Behavior Order</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-12 4</u></a>	Write to Arbitrary Storage Location	<a href="#"><u>CWE-123: Write-what-where Condition</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-12 3</u></a>	Requirement Violation	<a href="#"><u>CWE-573: Improper Following of Specification by Caller</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-12 2</u></a>	Lack of Proper Signature Verification	<a href="#"><u>CWE-345: Insufficient Verification of Data Authenticity</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-12 1</u></a>	Missing Protection against Signature Replay Attacks	<a href="#"><u>CWE-347: Improper Verification of Cryptographic Signature</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-12 0</u></a>	Weak Sources of Randomness from Chain Attributes	<a href="#"><u>CWE-330: Use of Insufficiently Random Values</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-11 9</u></a>	Shadowing State Variables	<a href="#"><u>CWE-710: Improper Adherence to Coding Standards</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-11 8</u></a>	Incorrect Constructor Name	<a href="#"><u>CWE-665: Improper Initialization</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-11 7</u></a>	Signature Malleability	<a href="#"><u>CWE-347: Improper Verification of Cryptographic Signature</u></a>	<b>PASSED</b>

<a href="#">SW C-11 6</a>	Timestamp Dependence	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	<b>PASSED</b>
<a href="#">SW C-11 5</a>	Authorization through tx.origin	<a href="#">CWE-477: Use of Obsolete Function</a>	<b>PASSED</b>
<a href="#">SW C-11 4</a>	Transaction Order Dependence	<a href="#">CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</a>	<b>PASSED</b>
<a href="#">SW C-11 3</a>	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	<b>PASSED</b>
<a href="#">SW C-11 2</a>	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	<b>PASSED</b>
<a href="#">SW C-11 1</a>	Use of Deprecated Solidity Functions	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	<b>PASSED</b>
<a href="#">SW C-11 0</a>	Assert Violation	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	<b>PASSED</b>
<a href="#">SW C-10 9</a>	Uninitialized Storage Pointer	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	<b>PASSED</b>
<a href="#">SW C-10 8</a>	State Variable Default Visibility	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	<b>PASSED</b>
<a href="#">SW C-10 7</a>	Reentrancy	<a href="#">CWE-284: Improper Access Control</a>	<b>PASSED</b>

<a href="#"><u>SW C-10 5</u></a>	Unprotected Ether Withdrawal	<a href="#"><u>CWE-284: Improper Access Control</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-10 4</u></a>	Unchecked Call Return Value	<a href="#"><u>CWE-252: Unchecked Return Value</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-10 3</u></a>	Floating Pragma	<a href="#"><u>CWE-664: Improper Control of a Resource Through its Lifetime</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-10 2</u></a>	Outdated Compiler Version	<a href="#"><u>CWE-937: Using Components with Known Vulnerabilities</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-10 1</u></a>	Integer Overflow and Underflow	<a href="#"><u>CWE-682: Incorrect Calculation</u></a>	<b>PASSED</b>
<a href="#"><u>SW C-10 0</u></a>	Function Default Visibility	<a href="#"><u>CWE-710: Improper Adherence to Coding Standards</u></a>	<b>PASSED</b>



# Valid Proof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC**