BLOCK AUDIT REPORT

Smart Contract Security Audit Report

Meta Ninja (NINJA)

Block Audit Report Team received the Metaninja.sol file for smart contract security audit of the Meta Ninja (NINJA) on February 10, 2022. The following are the details and results of this smart contract security audit:

Project Name: Meta Ninja (NINJA)

The Contract address: 0xebf963a8f11631944de83e2d18773454ef45ae6c

Link Address:
https://bscscan.com/address/0xebf963a8f11631944de83e2d18773454ef45ae6c

The audit items and results:

(Other undiscovered security vulnerabilities are not included in the audit responsibility scope)

Audit Result: Passed

Audit Number: BAR0018010022022

Audit Date: February 10, 2022

Audit Team: Block Audit Report Team

Table of Content

Introduction

This Audit Report mainly focuses on the extensive security of Meta Ninja (NINJA) Smart Contract. With this report, we attempt to ensure the reliability and correctness of the smart contract by complete and rigorous assessment of the system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Block Audit Report team has performed rigorous testing of the project including the analysis of the code design patterns where we reviewed the smart contract architecture to ensure it is structured along with the safe use of standard inherited contracts and libraries. Our team also conducted a formal line by line inspection of the Smart Contract i.e., a manual review, to find potential issues including but not limited to;

- Race conditions
- Zero race conditions approval attacks
- Re-entrancy
- Transaction-ordering dependence
- Timestamp dependence
- Check-effects-interaction pattern (optimistic accounting)
- Decentralized denial-of-service attacks
- Secure ether transfer pattern
- Guard check pattern
- Fail-safe mode
- Gas-limits and infinite loops
- Call Stack depth

In the Unit testing Phase, we coded/conducted custom unit tests written against each function in the contract to verify the claimed functionality from our client.
In Automated Testing, we tested the Smart Contract with our standard set of multifunctional tools to identify vulnerabilities and security flaws.
The code was tested in collaboration of our multiple team members and this included but not limited to;

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analyzing the complexity of the code in depth and detailed, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

Audit Details

Project Name: Meta Ninja (NINJA)
Website/ Bscscan Code (**Mainnet**):
0xebf963a8F11631944dE83E2D18773454EF45aE6C

Languages: Solidity (Smart contract)
Platforms and Tools: Remix IDE, Truffle, Ganache, Mythril, Contract Library, Slither, Dapp.Tools, Echidna, Etheno

Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient and working according to the specifications. The audit activities can be grouped in the following three categories:

Security Sight

Identifying security related issues within each contract and the system of contract.

Sound Architecture

Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices, standard software design principle, design patterns and practices.

Code Correctness and Quality

A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Usability vs Security
- Sections of code with high complexity
- Quantity and quality of test coverage

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical Severity Issues

Issues of this level are critical to the smart contract's performance/functionality and should be fixed before moving to a production environment.

High level severity issues

Issues on this level are strongly suggested by the team to be fixed before moving to the production environment.

Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low level severity issues

Issues on this level are minor details and warning's that can remain unfixed but would be better fixed at some point in the future.

Issues Checking Status

№	Issue description	Checking status
1	Compiler warnings.	Passed
2	Race conditions and Reentrancy. Cross-function race conditions.	Passed
3	Oracle calls.	Passed
4	Timestamp dependence.	Passed
5	DoS with Revert.	Passed
6	DoS with block gas limit.	Passed
7	Methods execution permissions.	Passed
8	Economy model.	Passed
9	The impact of the exchange rate on the logic.	Passed
10	Malicious Event log.	Passed
11	Scoping and Declarations.	Passed
12	Uninitialized storage pointers.	Passed
13	Arithmetic Operations accuracy.	Passed
14	Design Logic.	Passed
15	Cross-function race conditions.	Passed
16	Safe usage for Open Zeppelin module.	Passed
17	Fallback function security.	Passed
18	Send & receive ether.	Passed
19	Zero race condition approval attacks.	Passed
20	Short address attack.	Passed
21	Owner’s authority to freeze.	Passed
22	Attempt to block ether flows.	Passed
23	Redundant inheritance check.	Passed
24	Silent overrides of mapping structs.	Passed
25	Function state mutability.	Passed
26	Unnecessary conversion of type.	Passed

Used Code from other Framework/Smart Contracts (direct import)

[+] library SafeMath

- tryAdd(uint256 a, uint256 b)
- trySub(uint256 a, uint256 b)
- tryMul(uint256 a, uint256 b)
- tryDiv(uint256 a, uint256 b)
- tryMod(uint256 a, uint256 b)
- add(uint256 a, uint256 b)
- sub(uint256 a, uint256 b)
- mul(uint256 a, uint256 b)
- div(uint256 a, uint256 b)
- mod(uint256 a, uint256 b)
- sub(uint256 a, uint256 b, st ...
- div(uint256 a, uint256 b, st ...
- mod(uint256 a, uint256 b, st ...

[+] interface IBEP20

- totalSupply()
- decimals()
- symbol()
- name()
- getOwner()
- balanceOf(address account)
- transfer(address recipient, ...
- allowance(address _owner, ad ...
- approve(address spender, uin ...
- transferFrom(address sender, ...
- authorize(address adr)
- unauthorize(address adr)
- isOwner(address account)
- isAuthorized(address adr)
- transferOwnership(address pa ...

[+] interface IDEXFactory

- createPair(address tokenA, a ...

[+] interface IDEXRouter

- factory()

- WETH()
- addLiquidity(
- addLiquidityETH(
- swapExactTokensForTokensSupp ...
- swapExactETHForTokensSupport ...
- swapExactTokensForETHSupport ...

[+] interface IDividendDistributor

- setDistributionCriteria(uint ...
- setShare(address shareholder ...
- deposit()
- process(uint256 gas)

[+] contract DividendDistributor is ID ...

- setDistributionCriteria(uint ...
- setShare(address shareholder ...
- deposit()
- process(uint256 gas)
- shouldDistribute(address sha ...
- distributeDividend(address s ...
- claimDividend()
- getUnpaidEarnings(address sh ...
- getCumulativeDividends(uint2 ...
- addShareholder(address share ...
- removeShareholder(address sh ...

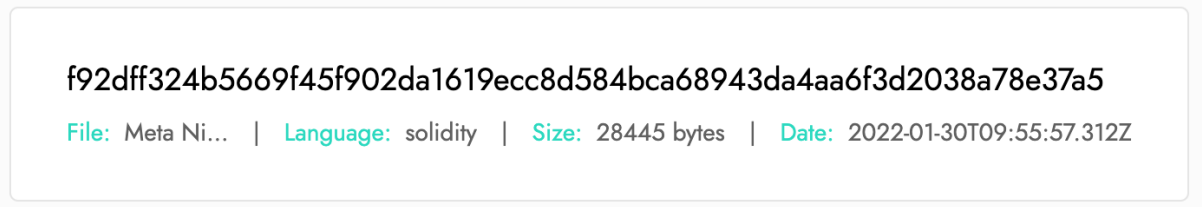
[+] contract MetaNinja is IBEP20, Auth *

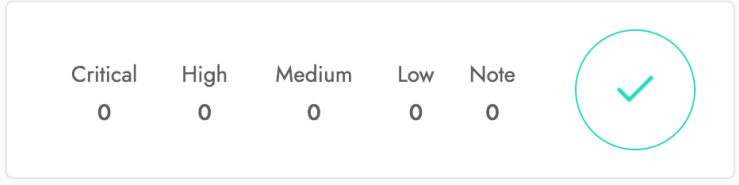
- totalSupply()
- decimals()
- symbol()
- name()
- getOwner()
- balanceOf(address account)
- allowance(address holder, ad ...
- approve(address spender, uin ...
- approveMax(address spender)
- transfer(address recipient, ...
- transferFrom(address sender, ...
- _transferFrom(address sender ...
- checkTxLimit(address sender, ...

- shouldTakeFee(address sender ...
- getTotalFee(bool selling)
- getMultipliedFee()
- takeFee(address sender, addr ...
- shouldSwapBack()
- swapBack()
- shouldAutoBuyback()
- triggerZeusBuyback(uint256 a ...
- clearBuybackMultiplier()
- triggerAutoBuyback()
- buyTokens(uint256 amount, ad ...
- setAutoBuybackSettings(bool ...
- setBuybackMultiplierSettings ...
- launched()
- launch()
- setTxLimit(uint256 amount)
- setIsDividendExempt(address ...
- setIsFeeExempt(address holde ...
- setIsTxLimitExempt(address h ...
- setFees(uint256 _liquidityFe ...
- setFeeReceivers(address _aut ...
- setSwapBackSettings(bool _en ...
- setTargetLiquidity(uint256 _ ...
- setDistributionCriteria(uint ...
- setDistributorSettings(uint2 ...
- getCirculatingSupply()
- getLiquidityBacking(uint256 ...
- isOverLiquified(uint256 targ ...

Manual Audit:

For this section the code was tested/read line by line by our auditors. We used Remix IDE's JavaScript VM and testnet Kovan to test the contract functionality in a simulated environment.





Critical Severity Issues

No critical severity issues found.

High Severity Issues

No high severity issues found.

Medium Severity Issues

No medium severity issues found.

Low Severity Issues

No low severity issues found.

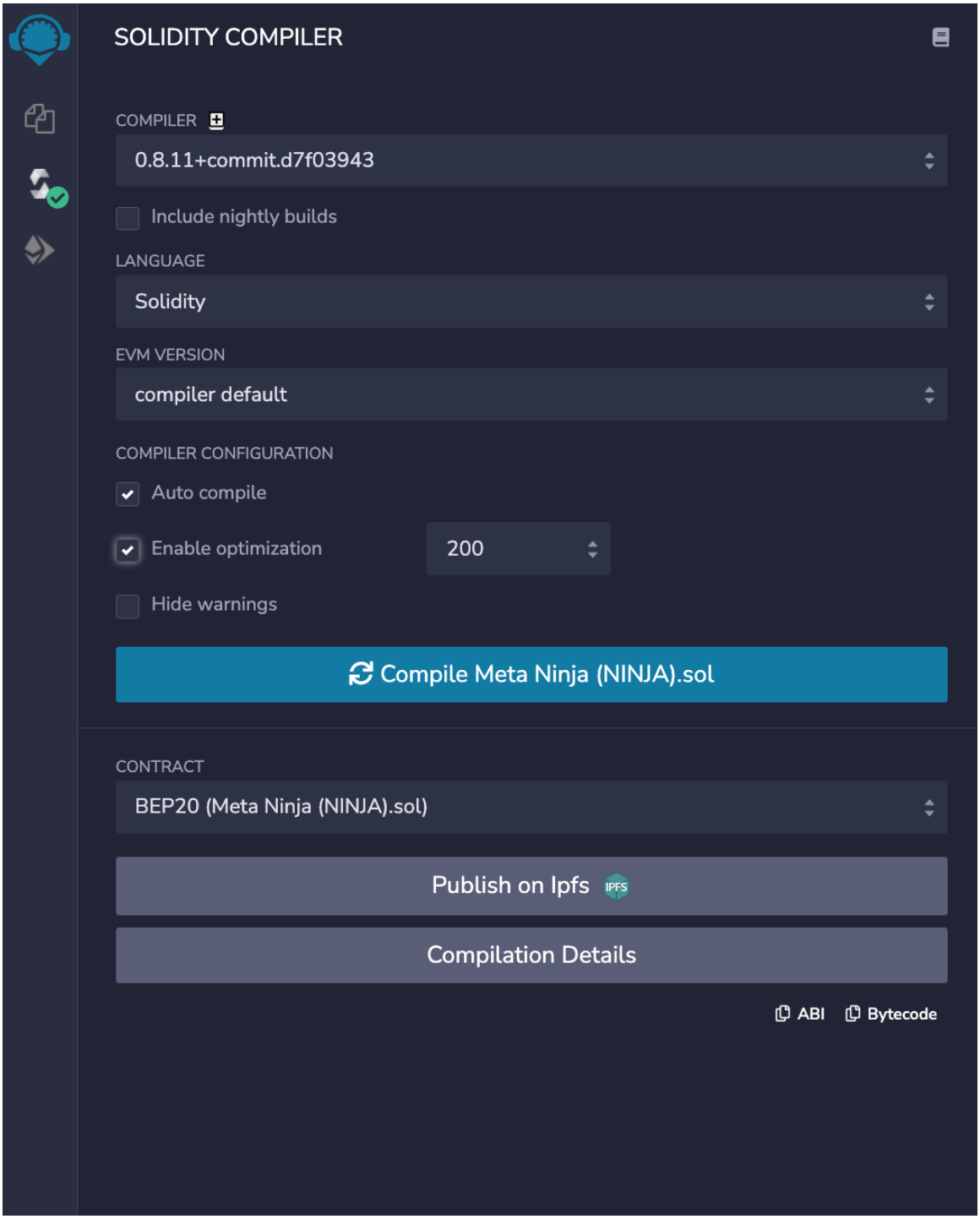
Owner privileges

- None

Automated Audit

Remix Compiler Warnings

It throws warnings by Solidity's compiler. If it encounters any errors the contract cannot be compiled and deployed.



Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for the client to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that the client should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for the client to conduct the client's own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, the client agrees to the terms of this disclaimer. If the client does not agree to the terms, then please immediately cease reading this report, and delete and destroy any/all copies of this report downloaded and/or printed by the client. This report is provided for information purposes only and stays on a non-reliance basis, and does not constitute investment advice. No one/ NONE shall have any rights to rely on the report or its contents, and BlockAudit and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives).
(BlockAudit) owes no duty of care towards the client or any other person, nor does BlockAudit claim any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and BlockAudit hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effects in relation to the report. Except and only to the extent that it is prohibited by law, BlockAudit hereby excludes all liability and responsibility, and neither the client nor any other person shall have any claim against BlockAudit, for any amount or kind of loss or damage that may result to the client or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the received smart contracts alone. No related/third-party smart contracts, applications or operations were reviewed for security. No product code has been reviewed.

Summary

Smart contracts received in file named: “Meta Ninja (NINJA)” do not contain any high severity issues!

Note:
Please read the disclaimer above and note, the audit claims NO statements or warranties on business model, investment advice/ attractiveness or code sustainability. This report is provided for the only set of contracts mentioned in the report and does not claim responsibility to include security audits for any other contracts deployed by Owner.

BLOCK AUDIT REPORT

Official Website
www.blockaudit.report

E-Mail
team@blockaudit.report

Twitter
https://twitter.com/BlockAudit

Github
https://github.com/blockauditreport