Meta Ninja

SMART CONTRACT AUDIT REPORT

Prepared by:

BlockAudit

Date Of Enrollment:

February 05th, 2022 - February 10th, 2022

Visit : www.blockaudit.report

TABLE OF CONTENTS

SUMMARY

This Audit Report mainly focuses on the extensive security of **Meta Ninja** Smart Contracts. 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.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified inpublic;
- Provide more transparency on privileged activities once the protocol is live.

OVERVIEW

Project Summary

Project Name	Meta Ninja
Logo	
Platform	BSC
Language	Solidity
Code Link	https://bscscan.com/address/0xebf963a8f11631944de83e2d18773454ef45ae6c

File Summary

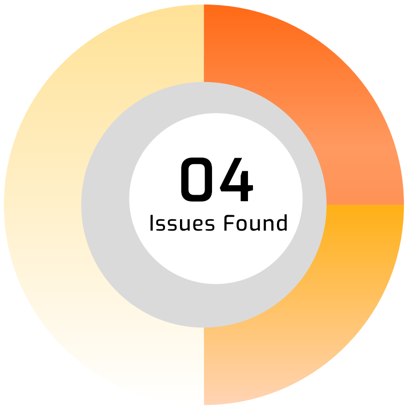
ID	File Name	Audit Status
NINJA	NINJA-MetaNinja.sol	Failed

Audit Summary

Date of Delivery	10th Feb 2022
Audit Methodology	Code Analysis. Automatic Assesment, Manual Review
Audit Result	Failed ✕
Audit Team	BlockAudit Report Team

FINDINGS

Critical	0	0.0%
High	1	25.00%
Medium	1	25.00%
Low	2	50.00%
Informational	0	0.0%
Ownership	0	0.0%
Gas Optimization	0	0.0%



Vulnerability Findings Summary

ID	Type	Line	Severity	Status
NINJA01	Reentrancy	955-1025	High	Reported
NINJA02	Divide before multiply	732 / 1043-1057 / 1169		Reported
NINJA03	Missing Events For Critical Parameters	836-839 / 841-844 / 846-849 / 888 / 1129 / 1188 / 1192 / 1196	Low	Reported
NINJA04	Missing Zero Address Validation	816 / 833	Low	Reported

NINJA01

Type	Reentracy
Severity	■ High
File	MetaNinja.sol
Line	810-880
Status	**Reported**

Description

_transfer(address, address,uint256) performs state variable changes after external calls, making it a potential target for a reentrancy attack if the inputs are malicious contracts.

Remediation

Update all bookkeeping state variables before transferring execution to an external contract or use Openzeppelin's ReentrancyGuard.

Snapshot:

```
function _transfer(address from, address to, uint256 amount) internal open(from, to) override
{…
}
```

NINJA02

Type	Divide before multiply
Severity	Medium Severity
File	MetaNinja.sol
Line	732 / 1043-1057 / 1169
Status	**Reported**

Description

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Remediation

In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

Snapshot

```
uint256 public _maxTxAmount = _totalSupply.div(1000).mul(5);
```

```
function swapAndLiquify(uint256 tokens) private
{
    uint256 swapableFee = liquidityFee.add(buybackFee);
    uint256 halfLiquidityTokens = tokens.div(swapableFee).mul(liquidityFee).div(2);
    uint256 swapableTokens = tokens.sub(halfLiquidityTokens);
    uint256 initialBalance = address(this).balance;
    swapTokensForEth(swapableTokens); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered
    // how much ETH did we just swap into?
    uint256 newBalance = address(this).balance.sub(initialBalance);
    uint256 bnbForLiquidity = newBalance.div(swapableFee).mul(liquidityFee).div(2);
    // add liquidity to uniswap
    addLiquidity(halfLiquidityTokens, bnbForLiquidity);
    emit SwapAndLiquify(halfLiquidityTokens, bnbForLiquidity, halfLiquidityTokens);
    //remaining bnb left for buyback purpose.
}
```

```
function checkForBuyBack() private
{
    uint256 balance = address(this).balance;
    if (buyBackEnabled && balance > minBalanceForBuyback)
    {
        if (balance > buyBackUpperLimit) { balance = buyBackUpperLimit; }
        buyBackTokens(balance.div(100).mul(buybackFactor));
    }
}
```

NINJA03

Type	Missing events for critical parameters
Severity	Low
File	MetaNinja.sol
Line	836-839 / 841-844 / 846-849 / 888 / 1129 / 1188 / 1192 / 1196
Status	**Reported**

Description

the functions below do not emit an event so it is hard to keep track of those critical parameters.

Remediation

Emit an event for critical parameter changes.

Snapshot

```
function setBUSDRewardsFee(uint256 value) external onlyOwner{
    BUSDRewardsFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);
}

function setLiquiditFee(uint256 value) external onlyOwner{
    liquidityFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);
}

function setMarketingFee(uint256 value) external onlyOwner{
    marketingFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);
}
```

```
function setSwapTokensAtAmount(uint256 _value) external onlyOwner
{
    swapTokensAtAmount = _value;
}
```

```
function setMaxTxAmount(uint256 _amount) external onlyOwner
{
    _maxTxAmount = _amount;
}
```

```
function setBuybackUpperLimit(uint256 buyBackLimit) external onlyOwner() {
    buyBackUpperLimit = buyBackLimit * 10**18;
}

function setMinBalanceForBuyback(uint256 _balanceInWei) external onlyOwner() {
    minBalanceForBuyback = _balanceInWei;
}

function setBuybackFactor(uint256 _percent) external onlyOwner() {
    buybackFactor = _percent;
}
```

NINJA04

Type	Missing zero address validation
Severity	Low
File	MetaNinja.sol
Line	943
Status	**Reported**

Description

setMarketingAddress(address) AND updateUniswapV2Router(address) do not check if wallet AND _uniswapV2Pair are zero addresses, So if Bob inputs zero addresses. funds can be lost.

Remediation

Check for zero address validation

Snapshot

```
function setMarketingWallet(address payable wallet) external onlyOwner{
    _marketingWalletAddress = wallet;
}
```

```
function updateUniswapV2Router(address newAddress) public onlyOwner {
    require(newAddress != address(uniswapV2Router), "MetaNinja: The router already has that address");
    emit UpdateUniswapV2Router(newAddress, address(uniswapV2Router));
    uniswapV2Router = IUniswapV2Router02(newAddress);
    address _uniswapV2Pair = IUniswapV2Factory(uniswapV2Router.factory())
        .createPair(address(this), uniswapV2Router.WETH());
    uniswapV2Pair = _uniswapV2Pair;
}
```

APPENDIX

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 in detail line-by-line manual review of the code.
- 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.

Issue Categories:

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

Critical Severity Issues

Issues of Critical Severity leaves smart contracts vulnerable to major exploits and can lead to asset loss and data loss. These can have significant impact on the functionality/performance of the smart contract.
We recommend these issues must be fixed before proceeding to MainNet..

High Severity Issues

Issues of High Severity are not as easy to exploit but they might endanger the execution of the smart contract and potentially create crucial problems.
Fixing these issues is highly recommended before proceeding to MainNet.

Medium Severity Issues

Issues on this level are not a major cause of vulnerability to the smart contract, they cannot lead to data-manipulations or asset loss but may affect funtionality.
It is important to fix these issues before proceeding to MainNet.

Low Severity Issues

Issues at this level are very low in their impact on the overall functionality and execution of the smart contract. These are mostly code-level violations or improper formatting.
These issues can be remain unfixed or can be fixed at a later date if the code is redeployed or forked.

Informational Findings

These are finding that our team comes accross when manually reviewing a smart contract which are important to know for the owners as well as users of a contract.
These issues must be acknowledged by the owners before we publish our report.

Ownership Priviledges

Owner of a smart contract can include certain rights and priviledges while deploying a smart contract that might be hidden deep inside the codebase and may make the project vulnerable to rug-pulls or other types of scams.
We at BlockAudit believe in transparency and hence we showcase Ownership priviledges separately so the owner as well as the investors can get a better understanding about the project.

Gas Optimization

Solidity gas optimization is the process of lowering the cost of operating your Solidity smart code. The term "gas" refers to the level of processing power required to perform specific tasks on the Ethereum network.
Each Ethereum transaction costs a fee since it requires the use of computer resources. It will deduct a fee anytime any function in the smart contract is invoked by the contract's owner or users.

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.

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 juriNINJAiction) 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.

Note: The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **Meta Ninja**
team put a bug bounty program in place to encourage further analysis of the smart contracts by other third parties

About BlockAudit

BlockAudit is an industry leading security organisation that helps web3 blockchain based projects with their security and correctness of their smart-contracts. With years of experience we have a dedicated team that is capable of performing audits in a wide variety of languages including HTML, PHP, JS, Node, React, Native, Solidity, Rust and other Web3 frameworks for DApps, DeFi, GameFi and Metaverse platforms.

With a mission to make web3 a safe and secure place BlockAudit is commited to provide it's partners with a budget and investor friendly security Audit Report that will increase the value of their projects significantly.

www.blockaudit.report

team@blockaudit.report

@BlockAudit

github.com/Block-Audit-Report