



**JAIHO**

**Smart Contract Review**

**Deliverable: Smart Contract Re-Audit Report**

**Security Report**

**September 2021**

## Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out in this publication are elaborated in the specific for only project.

eNebula Solutions does not guarantee the authenticity of the project or organization or team of members that is connected/owner behind the project or nor accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

eNebula Solutions retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

© eNebula Solutions, 2021.

## Report Summary

<b>Title</b>	JAIHO Smart Contract Audit		
<b>Project Owner</b>	JAIHO		
<b>Type</b>	Public		
<b>Reviewed by</b>	Vatsal Raychura	<b>Revision date</b>	13/09/2021
<b>Approved by</b>	eNebula Solutions Private Limited	<b>Approval date</b>	13/09/2021
		<b>Nº Pages</b>	<b>30</b>

## Overview

### Background

JAIHO requested that eNebula Solutions perform an Extensive Smart Contract audit of their Smart Contract.

### Project Dates

The following is the project schedule for this review and report:

- **September 12:** Smart Contract Review Completed (*Completed*)
- **September 12:** Delivery of Smart Contract Audit Report (*Completed*)
- **September 13:** Delivery of Smart Contract Re-Audit Report (*Completed*)

### Review Team

The following eNebula Solutions team member participated in this review:

- Sejal Barad, Security Researcher and Engineer
- Vatsal Raychura, Security Researcher and Engineer

## Coverage

### Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of JAIHO.

The following documentation repositories were considered in-scope for the review:

- JAIHO Project:  
<https://bscscan.com/address/0x2fd2799e83a723b19026a979899dfb70bbf6bf6b#code>

## Introduction

Given the opportunity to review JAIHO Project's smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is ready to launch after resolving the mentioned issues, there are no critical or high issues found related to business logic, security or performance.

About JAIHO: -

Item	Description
Issuer	JAIHO
Website	<a href="https://www.jaihocrypto.com/">https://www.jaihocrypto.com/</a>
Type	BEP20
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	September 13, 2021

The Test Method Information: -

Test method	Description
<b>Black box testing</b>	Conduct security tests from an attacker's perspective externally.
<b>Grey box testing</b>	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
<b>White box testing</b>	Based on the open-source code, non-open-source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

# Smart Contract Audit

The vulnerability severity level information:

Level	Description
<b>Critical</b>	Critical severity vulnerabilities will have a significant effect on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
<b>High</b>	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
<b>Medium</b>	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
<b>Low</b>	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
<b>Weakness</b>	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

The Full List of Check Items:

Category	Check Item
<b>Basic Coding Bugs</b>	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	MONEY-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
Transaction Ordering Dependence	
Deprecated Uses	
<b>Semantic Consistency Checks</b>	Semantic Consistency Checks
	Business Logics Review

# Smart Contract Audit

<b>Advanced DeFi Scrutiny</b>	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
<b>Additional Recommendations</b>	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

Category	Summary
<b>Configuration</b>	Weaknesses in this category are typically introduced during the configuration of the software.
<b>Data Processing Issues</b>	Weaknesses in this category are typically found in functionality that processes data.
<b>Numeric Errors</b>	Weaknesses in this category are related to improper calculation or conversion of numbers.
<b>Security Features</b>	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
<b>Time and State</b>	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
<b>Error Conditions, Return Values, Status Codes</b>	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
<b>Resource Management</b>	Weaknesses in this category are related to improper management of system resources.

## Smart Contract Audit

<b>Behavioral Issues</b>	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
<b>Business Logics</b>	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
<b>Initialization and Cleanup</b>	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
<b>Arguments and Parameters</b>	Weaknesses in this category are related to improper use arguments or parameters within function calls.
<b>Expression Issues</b>	Weaknesses in this category are related to incorrectly written expressions within code.
<b>Coding Practices</b>	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.



## Findings

### Summary

Here is a summary of our findings after analyzing the JAIHO's Smart Contract. During the first phase of our audit, we studied the smart contract sourcecode and ran our in-house static code analyzer through the Specific tool. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. We further manually review businesslogics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	No. of Issues
Critical	0
High	0
Medium	0
Low	3
Total	3

We have so far identified that there are potential issues with severity of **0 Critical, 0 High, 0 Medium, and 3 Low**. Overall, these smart contracts are well- designed and engineered, though the implementation can be improved and bug free by common recommendations given under POCs.

## Functional Overview

(\$) = payable function	[Pub] public
# = non-constant function	[Ext] external
	[Prv] private
	[Int] internal

- + [Int] IERC20
  - [Ext] totalSupply
  - [Ext] balanceOf
  - [Ext] transfer #
  - [Ext] allowance
  - [Ext] approve #
  - [Ext] transferFrom #
  
- + [Lib] SafeMath
  - [Int] add
  - [Int] sub
  - [Int] sub
  - [Int] mul
  - [Int] div
  - [Int] div
  - [Int] mod
  - [Int] mod
  
- + Context
  - [Int] \_msgSender
  - [Int] \_msgData

- + [Lib] Address
  - [Int] isContract
  - [Int] sendValue #
  - [Int] functionCall #
  - [Int] functionCall #
  - [Int] functionCallWithValue #
  - [Int] functionCallWithValue #
  - [Prv] \_functionCallWithValue #

- + Ownable (Context)
  - [Pub] <Constructor> #
  - [Pub] owner
  - [Pub] renounceOwnership #
    - modifiers: onlyOwner
  - [Pub] transferOwnership #
    - modifiers: onlyOwner
  - [Pub] geUnlockTime
  - [Pub] lock #
    - modifiers: onlyOwner
  - [Pub] unlock #

- + [Int] IUniswapV2Factory
  - [Ext] feeTo
  - [Ext] feeToSetter
  - [Ext] getPair
  - [Ext] allPairs
  - [Ext] allPairsLength
  - [Ext] createPair #
  - [Ext] setFeeTo #
  - [Ext] setFeeToSetter #

```
+ [Int] IUniswapV2Pair
- [Ext] name
- [Ext] symbol
- [Ext] decimals
- [Ext] totalSupply
- [Ext] balanceOf
- [Ext] allowance
- [Ext] approve #
- [Ext] transfer #
- [Ext] transferFrom #
- [Ext] DOMAIN_SEPARATOR
- [Ext] PERMIT_TYPEHASH
- [Ext] nonces
- [Ext] permit #
- [Ext] MINIMUM_LIQUIDITY
- [Ext] factory
- [Ext] token0
- [Ext] token1
- [Ext] getReserves
- [Ext] price0CumulativeLast
- [Ext] price1CumulativeLast
- [Ext] kLast
- [Ext] mint #
- [Ext] burn #
- [Ext] swap #
- [Ext] skim #
- [Ext] sync #
- [Ext] initialize #

+ [Int] IUniswapV2Router01
- [Ext] factory
```

# Smart Contract Audit

- [Ext] WETH
- [Ext] addLiquidity #
- [Ext] addLiquidityETH (\$)
- [Ext] removeLiquidity #
- [Ext] removeLiquidityETH #
- [Ext] removeLiquidityWithPermit #
- [Ext] removeLiquidityETHWithPermit #
- [Ext] swapExactTokensForTokens #
- [Ext] swapTokensForExactTokens #
- [Ext] swapExactETHForTokens (\$)
- [Ext] swapTokensForExactETH #
- [Ext] swapExactTokensForETH #
- [Ext] swapETHForExactTokens (\$)
- [Ext] quote
- [Ext] getAmountOut
- [Ext] getAmountIn
- [Ext] getAmountsOut
- [Ext] getAmountsIn
  
- + [Int] IUniswapV2Router02 (IUniswapV2Router01)
  - [Ext] removeLiquidityETHSupportingFeeOnTransferTokens #
  - [Ext] removeLiquidityETHWithPermitSupportingFeeOnTransferTokens #
  - [Ext] swapExactTokensForTokensSupportingFeeOnTransferTokens #
  - [Ext] swapExactETHForTokensSupportingFeeOnTransferTokens (\$)
  - [Ext] swapExactTokensForETHSupportingFeeOnTransferTokens #
  
- + JaiHo (Context, IERC20, Ownable)
  - [Pub] <Constructor> #
  - [Pub] name
  - [Pub] symbol
  - [Pub] decimals

- [Pub] totalSupply
- [Pub] balanceOf
- [Pub] transfer #
- [Pub] allowance
- [Pub] approve #
- [Pub] transferFrom #
- [Pub] increaseAllowance #
- [Pub] decreaseAllowance #
- [Pub] isExcludedFromReward
- [Pub] totalFees
- [Pub] deliver #
- [Pub] reflectionFromToken
- [Pub] tokenFromReflection
- [Pub] excludeFromReward #
  - modifiers: onlyOwner
- [Ext] includeInReward #
  - modifiers: onlyOwner
- [Prv] \_transferBothExcluded #
- [Ext] <Fallback> (\$)
- [Prv] \_reflectFee #
- [Prv] \_getValues
- [Prv] \_getTValues
- [Prv] \_getRValues
- [Prv] \_getRate
- [Prv] \_getCurrentSupply
- [Prv] \_takeLiquidity #
- [Prv] calculateTaxFee
- [Prv] calculateLiquidityFee
- [Prv] removeAllFee #
- [Prv] restoreAllFee #
- [Pub] isExcludedFromFee

- [Prv] \_approve #
- [Prv] \_transfer #
- [Prv] swapAndLiquify #
  - modifiers: lockTheSwap
- [Prv] swapTokensForEth #
- [Prv] addLiquidity #
- [Prv] \_tokenTransfer #
- [Prv] \_transferStandard #
- [Prv] \_transferToExcluded #
- [Prv] \_transferFromExcluded #
- [Pub] excludeFromFee #
  - modifiers: onlyOwner
- [Pub] includeInFee #
  - modifiers: onlyOwner
- [Ext] enableAllFees #
  - modifiers: onlyOwner
- [Ext] setCharityWallet #
  - modifiers: onlyOwner
- [Ext] SetNumTokensSellToAddToLiquidity #
  - modifiers: onlyOwner
- [Ext] setMaxTxAmount #
  - modifiers: onlyOwner
- [Pub] setSwapAndLiquifyEnabled #
  - modifiers: onlyOwner

## Detailed Results

### Issues Checking Status

#### 1. Floating Pragma

- SWC ID:103
- Severity: Low
- Location:  
<https://bscscan.com/address/0x2fd2799e83a723b19026a979899dfb70bbf6bf6b#code>
- Relationships: CWE-664: Improper Control of a Resource Through its Lifetime
- Description: A floating pragma is set. The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

```
21  
22  pragma solidity ^0.8.0;  
23
```

- Remediations: Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.
- Acknowledged: After the first phase of Audit, this issue was discussed with the JaiHo's dev team, and they Acknowledged this but as no serious or performance issue with this, they've decided to remain the code unchanged.



## 2. State Variable Default Visibility

- SWC ID:108
- Severity: Low
- Location:  
<https://bscscan.com/address/0x2fd2799e83a723b19026a979899dfb70bbf6bf6b#code>
- Relationships: State variable visibility is not set. It is best practice to set the visibility of state variables explicitly. The default visibility for "inSwapAndLiquify" is internal. Other possible visibility settings are public and private.

```
741
742     bool inSwapAndLiquify;
743     bool public swapAndLiquifyEnabled = false;
744
```

- Remediations: Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.
- Acknowledged: After the first phase of Audit, this issue was discussed with the JaiHo's dev team, and they Acknowledged this but as no serious or performance issue with this, they've decided to remain the code unchanged.

## 3. Block values as a proxy for time

- SWC ID:116
- Severity: Low
- Location:  
<https://bscscan.com/address/0x2fd2799e83a723b19026a979899dfb70bbf6bf6b#code>
- Relationships: CWE-829: Inclusion of Functionality from Untrusted Control Sphere
- Description: Here in function unlock() A control flow decision is made based on The 'block.timestamp' environment variable. Note that the values of variables like coinbase, gaslimit, block number, and timestamp are predictable and can be manipulated by malicious miners. Also, keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that the use of these variables introduces a certain level of trust into miners.

```
481     function unlock() public virtual {
482         require(_previousOwner == msg.sender, "You don't have permission to unlock")
483         require(block.timestamp < _lockTime , "Contract is locked until 7 days");
484         emit OwnershipTransferred(_owner, _previousOwner);
485         _owner = _previousOwner;
486     }
```

- Remediations: Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use of oracles.
- Acknowledged: After the first phase of Audit, this issue was discussed with the JaiHo's dev team, and they Acknowledged this but as no serious or performance issue with this, they've decided to remain the code unchanged.

# Smart Contract Audit

## Automated Tools Results

Slither: -

```
Reentrancy in Jaiho.transfer(address,address,uint256) (Jaiho.sol#994-1022):
  External calls:
  - swapAndLiquify(contractTokenBalance) (Jaiho.sol#1017)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,owner(),block.timestamp) (Jaiho.sol#1070-1077)
    - _uniswapV2Router.swapExactTokensForETHsupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (Jaiho.sol#1050-1062)
  External calls sending eth:
  - swapAndLiquify(contractTokenBalance) (Jaiho.sol#1017)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,owner(),block.timestamp) (Jaiho.sol#1070-1077)
  State variables written after the call(s):
  - _tokenTransfer(from,to,amount) (Jaiho.sol#1021)
    - _owned[address(this)] = _owned[address(this)].add(tLiquidity) (Jaiho.sol#951)
    - _owned[sender] = _owned[sender].sub(rAmount) (Jaiho.sol#1124)
    - _owned[sender] = _owned[sender].sub(rAmount) (Jaiho.sol#1131)
    - _owned[sender] = _owned[sender].sub(rAmount) (Jaiho.sol#1144)
    - _owned[sender] = _owned[sender].sub(rAmount) (Jaiho.sol#1154)
    - _owned[sender] = _owned[sender].sub(rAmount) (Jaiho.sol#1162)
    - _owned[recipient] = _owned[recipient].add(rTransferAmount) (Jaiho.sol#1124)
    - _owned[recipient] = _owned[recipient].add(rTransferAmount) (Jaiho.sol#1131)
    - _owned[recipient] = _owned[recipient].add(rTransferAmount) (Jaiho.sol#1143)
    - _owned[recipient] = _owned[recipient].add(rTransferAmount) (Jaiho.sol#1153)
  - _tokenTransfer(from,to,amount) (Jaiho.sol#1021)
    - rTotal = rTotal.sub(rFee) (Jaiho.sol#986)
  - _tokenTransfer(from,to,amount) (Jaiho.sol#1021)
    - _owned[address(this)] = _owned[address(this)].add(tLiquidity) (Jaiho.sol#951)
    - _owned[sender] = _owned[sender].sub(rAmount) (Jaiho.sol#1091)
    - _owned[sender] = _owned[sender].sub(rAmount) (Jaiho.sol#1143)
    - _owned[recipient] = _owned[recipient].add(rTransferAmount) (Jaiho.sol#1134)
    - _owned[recipient] = _owned[recipient].add(rTransferAmount) (Jaiho.sol#1101)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

Jaiho.addLiquidity(uint256,uint256) (Jaiho.sol#1065-1076) ignores return value by _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,owner(),block.timestamp) (Jaiho.sol#1070-1077)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Jaiho.allowance(address,address).owner (Jaiho.sol#887) shadows:
  - _Denable.owner() (Jaiho.sol#434-436) (function)
Jaiho._approve(address,address,uint256).owner (Jaiho.sol#98a) shadows:
  - _Denable.owner() (Jaiho.sol#434-436) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Jaiho.SetNumTokensSellToAddToLiquidity(uint256) (Jaiho.sol#1177-1179) should emit an event for:
  - numTokensSellToAddToLiquidity = newAmount (Jaiho.sol#1178)
Jaiho.setMaxTaxAmount(uint256) (Jaiho.sol#1181-1184) should emit an event for:
  - _maxTaxAmount = maxTaxAmount (Jaiho.sol#1183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Jaiho.setCharityWallet(address).newWallet (Jaiho.sol#1171) lacks a zero-check on :
  - charityWallet = newWallet (Jaiho.sol#1174)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in Jaiho.transfer(address,address,uint256) (Jaiho.sol#994-1022):
  External calls:
  - swapAndLiquify(contractTokenBalance) (Jaiho.sol#1017)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,owner(),block.timestamp) (Jaiho.sol#1070-1077)
    - _uniswapV2Router.swapExactTokensForETHsupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (Jaiho.sol#1050-1062)
  External calls sending eth:
  - swapAndLiquify(contractTokenBalance) (Jaiho.sol#1017)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,owner(),block.timestamp) (Jaiho.sol#1070-1077)
  State variables written after the call(s):
  - _tokenTransfer(from,to,amount) (Jaiho.sol#1021)
    - _burnFee = 1 (Jaiho.sol#970)
    - _burnFee = 0 (Jaiho.sol#971)
  - _tokenTransfer(from,to,amount) (Jaiho.sol#1021)
    - _charityFee = 1 (Jaiho.sol#973)
    - _charityFee = 0 (Jaiho.sol#972)
  - _tokenTransfer(from,to,amount) (Jaiho.sol#1021)
    - _liquidityFee = 0 (Jaiho.sol#970)
    - _liquidityFee = 4 (Jaiho.sol#977)
    - _liquidityFee = 0 (Jaiho.sol#1107)
    - _liquidityFee = _previousLiquidityFee (Jaiho.sol#1115)
  - _tokenTransfer(from,to,amount) (Jaiho.sol#1021)
    - _tFeeTotal = _tFeeTotal.add(tFee) (Jaiho.sol#967)
  - _tokenTransfer(from,to,amount) (Jaiho.sol#1021)
    - _taxFee = 0 (Jaiho.sol#969)
    - _taxFee = 2 (Jaiho.sol#970)
    - _taxFee = 0 (Jaiho.sol#1106)
    - _taxFee = _previousTaxFee (Jaiho.sol#1114)
```

# Smart Contract Audit

```
Reentrancy in Jaiho.constructor() (Jaiho.sol#762-779):
  External calls:
  - _uniswapV2Pair = _UniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this), _uniswapV2Router.WETH()) (Jaiho.sol#768-769)
  State variables written after the call(s):
  - _isExcludedFromFee[owner()] = true (Jaiho.sol#775)
  - _isExcludedFromFee[address(this)] = true (Jaiho.sol#776)
  - _uniswapV2Router = _uniswapV2Router (Jaiho.sol#772)
Reentrancy in Jaiho.swapAndLiquify(uint256) (Jaiho.sol#1024-1045):
  External calls:
  - swapTokensForEth(half) (Jaiho.sol#1036)
    - _uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount, 0, path, address(this), block.timestamp) (Jaiho.sol#1056)
-1062)
  - addLiquidity(otherHalf, newBalance) (Jaiho.sol#1042)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
  External calls sending eth:
  - addLiquidity(otherHalf, newBalance) (Jaiho.sol#1042)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
  State variables written after the call(s):
  - addLiquidity(otherHalf, newBalance) (Jaiho.sol#1042)
    - allowances[owner][spender] = amount (Jaiho.sol#990)
Reentrancy in Jaiho.transferFrom(address, address, uint256) (Jaiho.sol#816-826):
  External calls:
  - _transfer(sender, recipient, amount) (Jaiho.sol#817)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
    - _uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount, 0, path, address(this), block.timestamp) (Jaiho.sol#1056)
-1062)
  External calls sending eth:
  - _transfer(sender, recipient, amount) (Jaiho.sol#817)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
  State variables written after the call(s):
  - _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, ERC20: transfer amount exceeds allowance)) (Jaiho.sol#818)
    - allowances[owner][spender] = amount (Jaiho.sol#990)
Reference: https://github.com/cryptic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
Reentrancy in Jaiho.transfer(address, address, uint256) (Jaiho.sol#994-1022):
  External calls:
  - swapAndLiquify(contractTokenBalance) (Jaiho.sol#1017)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
    - _uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount, 0, path, address(this), block.timestamp) (Jaiho.sol#1056)
-1062)
  External calls sending eth:
  - swapAndLiquify(contractTokenBalance) (Jaiho.sol#1017)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
  Event emitted after the call(s):
  - Transfer(sender, recipient, tTransferAmount) (Jaiho.sol#1120)
    - _tokenTransfer(from, to, amount) (Jaiho.sol#1021)
  - Transfer(sender, recipient, tTransferAmount) (Jaiho.sol#1148)
    - _tokenTransfer(from, to, amount) (Jaiho.sol#1021)
  - Transfer(sender, recipient, tTransferAmount) (Jaiho.sol#1138)
    - _tokenTransfer(from, to, amount) (Jaiho.sol#1021)
  - Transfer(sender, recipient, tTransferAmount) (Jaiho.sol#897)
    - _tokenTransfer(from, to, amount) (Jaiho.sol#1021)
Reentrancy in Jaiho.constructor() (Jaiho.sol#762-779):
  External calls:
  - _uniswapV2Pair = _UniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this), _uniswapV2Router.WETH()) (Jaiho.sol#768-769)
  Event emitted after the call(s):
  - Transfer(address(0), _msgSender(), tTotal) (Jaiho.sol#778)
Reentrancy in Jaiho.swapAndLiquify(uint256) (Jaiho.sol#1024-1045):
  External calls:
  - swapTokensForEth(half) (Jaiho.sol#1036)
    - _uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount, 0, path, address(this), block.timestamp) (Jaiho.sol#1056)
-1062)
  - addLiquidity(otherHalf, newBalance) (Jaiho.sol#1042)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
  External calls sending eth:
  - addLiquidity(otherHalf, newBalance) (Jaiho.sol#1042)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
  Event emitted after the call(s):
  - Approval(owner, spender, amount) (Jaiho.sol#991)
    - addLiquidity(otherHalf, newBalance) (Jaiho.sol#1042)
  - SwapAndLiquify(half, newBalance, otherHalf) (Jaiho.sol#1044)
Reentrancy in Jaiho.transferFrom(address, address, uint256) (Jaiho.sol#816-826):
  External calls:
  - _transfer(sender, recipient, amount) (Jaiho.sol#817)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
    - _uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount, 0, path, address(this), block.timestamp) (Jaiho.sol#1056)
-1062)
  External calls sending eth:
  - _transfer(sender, recipient, amount) (Jaiho.sol#817)
    - _uniswapV2Router.addLiquidityETH(value: ethAmount)(address(this), tokenAmount, 0, 0, owner(), block.timestamp) (Jaiho.sol#1078-1077)
  Event emitted after the call(s):
  - Approval(owner, spender, amount) (Jaiho.sol#991)
    - approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, ERC20: transfer amount exceeds allowance)) (Jaiho.sol#818)
Reference: https://github.com/cryptic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
Ownable.unlock() (Jaiho.sol#481-486) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool, string)(block.timestamp < _lockTime, Contract is locked until 7 days) (Jaiho.sol#483)
Reference: https://github.com/cryptic/slither/wiki/Detector-Documentation#block-timestamp
Address.isContract(address) (Jaiho.sol#286-295) uses assembly
  - INLINE ASM (Jaiho.sol#293)
Address.functionCallWithValue(address, bytes, uint256, string) (Jaiho.sol#379-400) uses assembly
  - INLINE ASM (Jaiho.sol#392-395)
Reference: https://github.com/cryptic/slither/wiki/Detector-Documentation#assembly-usage
Address.functionCallWithValue(address, bytes, uint256, string) (Jaiho.sol#379-400) is never used and should be removed
Address.functionCall(address, bytes) (Jaiho.sol#339-341) is never used and should be removed
Address.functionCall(address, bytes, string) (Jaiho.sol#349-351) is never used and should be removed
Address.functionCallWithValue(address, bytes, uint256) (Jaiho.sol#364-366) is never used and should be removed
Address.functionCallWithValue(address, bytes, uint256, string) (Jaiho.sol#374-377) is never used and should be removed
Address.isContract(address) (Jaiho.sol#286-295) is never used and should be removed
Address.sendValue(address, uint256) (Jaiho.sol#313-319) is never used and should be removed
Context.msgData() (Jaiho.sol#250-261) is never used and should be removed
SafeMath.mod(uint256, uint256) (Jaiho.sol#231-235) is never used and should be removed
SafeMath.mod(uint256, uint256, string) (Jaiho.sol#247-250) is never used and should be removed
Reference: https://github.com/cryptic/slither/wiki/Detector-Documentation#dead-code
```

# Smart Contract Audit

```
JaiHo._total (JaiHo.sol#718) is set pre-construction with a non-constant function or state variable:  
- (MAX - (MAX % _total))  
JaiHo._previousTaxFee (JaiHo.sol#726) is set pre-construction with a non-constant function or state variable:  
- taxFee  
JaiHo._previousLiquidityFee (JaiHo.sol#729) is set pre-construction with a non-constant function or state variable:  
- liquidityFee  
JaiHo._previousBurnFee (JaiHo.sol#732) is set pre-construction with a non-constant function or state variable:  
- burnFee  
JaiHo._previousCharityFee (JaiHo.sol#737) is set pre-construction with a non-constant function or state variable:  
- charityFee  
Reference: https://github.com/crytic/sltther/wiki/Detector-Documentation#function-initializing-state  
  
Pragma version<0.8.0 (JaiHo.sol#422) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.0  
solc-0.8.0 is not recommended for deployment  
Reference: https://github.com/crytic/sltther/wiki/Detector-Documentation#incorrect-versions-of-solidity  
  
Low level call in Address.sendValue(address,uint256) (JaiHo.sol#319-319):  
- (success) = recipient.call{value: amount}() (JaiHo.sol#317)  
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (JaiHo.sol#379-400):  
- (success,returnedData) = target.call{value: weiValue}(data) (JaiHo.sol#383)  
Reference: https://github.com/crytic/sltther/wiki/Detector-Documentation#low-level-calls  
  
Function JaiHo.swapV2Pair,DBMAIN_SEPARATOR() (JaiHo.sol#525) is not in mixedCase  
Function JaiHo.swapV2Pair,PERMIT_TYPEHASH() (JaiHo.sol#526) is not in mixedCase  
Function JaiHo.swapV2Pair,MINIMUM_LIQUIDITY() (JaiHo.sol#543) is not in mixedCase  
Function JaiHo.swapV2Router01,WEETH() (JaiHo.sol#565) is not in mixedCase  
Parameter JaiHo.calculateTaxFee(uint256),_amount (JaiHo.sol#956) is not in mixedCase  
Parameter JaiHo.calculateLiquidityFee(uint256),_amount (JaiHo.sol#962) is not in mixedCase  
Function JaiHo.SetNumTokensSellToAddToLiquidity(uint256) (JaiHo.sol#1177-1179) is not in mixedCase  
Parameter JaiHo.setSwapAndLiquifyEnabled(bool),_enabled (JaiHo.sol#1186) is not in mixedCase  
Variable JaiHo._taxFee (JaiHo.sol#725) is not in mixedCase  
Variable JaiHo._liquidityFee (JaiHo.sol#728) is not in mixedCase  
Variable JaiHo._burnFee (JaiHo.sol#731) is not in mixedCase  
Variable JaiHo._charityFee (JaiHo.sol#735) is not in mixedCase  
Variable JaiHo._maxAmount (JaiHo.sol#745) is not in mixedCase  
Reference: https://github.com/crytic/sltther/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions  
  
Redundant expression "this (JaiHo.sol#259)" inContext (JaiHo.sol#253-262)  
Reference: https://github.com/crytic/sltther/wiki/Detector-Documentation#redundant-statements  
  
Variable JaiHo.swapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (JaiHo.sol#570) is too similar  
r to JaiHo.swapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (JaiHo.sol#571)  
Variable JaiHo._transferFromExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1142) is too similar to JaiHo._transferStandard(address,addre  
ss,uint256).tTransferAmount (JaiHo.sol#1123)  
Variable JaiHo._getValues(uint256).rTransferAmount (JaiHo.sol#912) is too similar to JaiHo._transferStandard(address,address,uint256).tTransferAmount  
(JaiHo.sol#1123)  
Variable JaiHo.reflectionFromToken(uint256,bool),rTransferAmount (JaiHo.sol#855) is too similar to JaiHo._transferStandard(address,address,uint256).tT  
ransferAmount (JaiHo.sol#1123)  
Variable JaiHo._transferToExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1132) is too similar to JaiHo._transferToExcluded(address,addre  
ss,uint256).tTransferAmount (JaiHo.sol#1132)  
Variable JaiHo._transferToExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1132) is too similar to JaiHo._transferFromExcluded(address,add  
ress,uint256).tTransferAmount (JaiHo.sol#1142)  
Variable JaiHo._getValues(uint256,uint256,uint256,uint256).rTransferAmount (JaiHo.sol#927) is too similar to JaiHo._getValues(uint256).tTransferAmou  
nt (JaiHo.sol#919)  
Variable JaiHo._transferToExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1132) is too similar to JaiHo._getValues(uint256).tTransferAmo  
unt (JaiHo.sol#919)  
Variable JaiHo._transferFromExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1142) is too similar to JaiHo._transferFromExcluded(address,a  
dress,uint256).tTransferAmount (JaiHo.sol#1142)  
Variable JaiHo._getValues(uint256,uint256,uint256,uint256).rTransferAmount (JaiHo.sol#927) is too similar to JaiHo._transferBothExcluded(address,add  
ress,uint256).tTransferAmount (JaiHo.sol#898)  
Variable JaiHo._transferToExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1132) is too similar to JaiHo._transferToExcluded(address,add  
ress,uint256).tTransferAmount (JaiHo.sol#898)  
Variable JaiHo._getValues(uint256,uint256,uint256,uint256).rTransferAmount (JaiHo.sol#927) is too similar to JaiHo._getValues(uint256).tTransferAmou  
nt (JaiHo.sol#911)  
Variable JaiHo._transferToExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1132) is too similar to JaiHo._getValues(uint256).tTransferAmou  
nt (JaiHo.sol#911)  
Variable JaiHo._transferBothExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#898) is too similar to JaiHo._getValues(uint256).tTransferAmo  
unt (JaiHo.sol#911)  
Variable JaiHo._transferFromExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1142) is too similar to JaiHo._getValues(uint256).tTransferA  
mount (JaiHo.sol#919)  
Variable JaiHo._transferBothExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#898) is too similar to JaiHo._transferBothExcluded(address,ad  
dress,uint256).tTransferAmount (JaiHo.sol#898)  
Variable JaiHo._getValues(uint256,uint256,uint256,uint256).rTransferAmount (JaiHo.sol#927) is too similar to JaiHo._transferStandard(address,address,  
uint256).tTransferAmount (JaiHo.sol#1123)  
Variable JaiHo._transferToExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1132) is too similar to JaiHo._transferStandard(address,address,  
uint256).tTransferAmount (JaiHo.sol#1123)  
Variable JaiHo._transferBothExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#898) is too similar to JaiHo._transferStandard(address,addre  
ss,uint256).tTransferAmount (JaiHo.sol#898)  
Variable JaiHo._transferFromExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1142) is too similar to JaiHo._transferBothExcluded(address,a  
dress,uint256).tTransferAmount (JaiHo.sol#898)  
Variable JaiHo._transferFromExcluded(address,address,uint256).rTransferAmount (JaiHo.sol#1142) is too similar to JaiHo._getValues(uint256).tTransferAm  
ount (JaiHo.sol#911)
```



# Smart Contract Audit

MythX: -

Line	SWC Title	Severity	Short Description
22	(SWC-103) Floating Pragma	Low	A floating pragma is set.
122	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
154	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
177	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
178	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
213	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "/" discovered
249	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "%" discovered
476	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "+" discovered
717	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "***" discovered
717	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
718	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "%" discovered
718	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
742	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
745	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "***" discovered
745	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
746	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "***" discovered
746	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "*" discovered
878	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
879	(SWC-110) Assert Violation	Unknown	Out of bounds array access
880	(SWC-101) Integer Overflow and Underflow	Unknown	Compiler-rewritable "<uint> - 1" discovered
880	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered
880	(SWC-110) Assert Violation	Unknown	Out of bounds array access
939	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
940	(SWC-110) Assert Violation	Unknown	Out of bounds array access
941	(SWC-110) Assert Violation	Unknown	Out of bounds array access
942	(SWC-110) Assert Violation	Unknown	Out of bounds array access
958	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "***" discovered
964	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "***" discovered
1050	(SWC-110) Assert Violation	Unknown	Out of bounds array access
1051	(SWC-110) Assert Violation	Unknown	Out of bounds array access





# Smart Contract Audit

Solhint: -

```
Lint results:

JaiHo.sol:22:1: Error: Compiler version ^0.8.0 does not satisfy the r semver requirement

JaiHo.sol:425:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)

JaiHo.sol:476:21: Error: Avoid to make time-based decisions in your business logic

JaiHo.sol:483:17: Error: Avoid to make time-based decisions in your business logic

JaiHo.sol:525:5: Error: Function name must be in mixedCase

JaiHo.sol:526:5: Error: Function name must be in mixedCase

JaiHo.sol:543:5: Error: Function name must be in mixedCase

JaiHo.sol:565:5: Error: Function name must be in mixedCase

JaiHo.sol:703:1: Error: Contract has 26 states declarations but allowed no more than 15

JaiHo.sol:742:5: Error: Explicitly mark visibility of state

JaiHo.sol:762:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)

JaiHo.sol:867:72: Error: Use double quotes for string literals

JaiHo.sol:903:32: Error: Code contains empty blocks

JaiHo.sol:1061:13: Error: Avoid to make time-based decisions in your business logic

JaiHo.sol:1076:13: Error: Avoid to make time-based decisions in your business logic

JaiHo.sol:1177:5: Error: Function name must be in mixedCase
```

## Basic Coding Bugs

### 1. Constructor Mismatch

- Description: Whether the contract name and its constructor are not identical to each other.
- Result: PASSED
- Severity: Critical

### 2. Ownership Takeover

- Description: Whether the set owner function is not protected.
- Result: PASSED
- Severity: Critical

### 3. Redundant Fallback Function

- Description: Whether the contract has a redundant fallback function.
- Result: PASSED
- Severity: Critical

### 4. Overflows & Underflows

- Description: Whether the contract has general overflow or underflow vulnerabilities
- Result: PASSED
- Severity: Critical

### 5. Reentrancy

- Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
- Result: PASSED
- Severity: Critical

### 6. MONEY-Giving Bug

- Description: Whether the contract returns funds to an arbitrary address.
- Result: PASSED
- Severity: High

## 7. Blackhole

- Description: Whether the contract locks ETH indefinitely: merely in without out.
- Result: PASSED
- Severity: High

## 8. Unauthorized Self-Destruct

- Description: Whether the contract can be killed by any arbitrary address.
- Result: PASSED
- Severity: Medium

## 9. Revert DoS

- Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
- Result: PASSED
- Severity: Medium

## 10. Unchecked External Call

- Description: Whether the contract has any external call without checking the return value.
- Result: PASSED
- Severity: Medium

## 11. Gasless Send

- Description: Whether the contract is vulnerable to gasless send.
- Result: PASSED
- Severity: Medium

## 12. Send Instead of Transfer

- Description: Whether the contract uses send instead of transfer.
- Result: PASSED
- Severity: Medium

## 13. Costly Loop

- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: PASSED
- Severity: Medium

## 14. (Unsafe) Use of Untrusted Libraries

- Description: Whether the contract use any suspicious libraries.
- Result: PASSED
- Severity: Medium

## 15. (Unsafe) Use of Predictable Variables

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: PASSED
- Severity: Medium

## 16. Transaction Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: PASSED
- Severity: Medium

## 17. Deprecated Uses

- Description: Whether the contract use the deprecated tx.origin to perform the authorization.
- Result: PASSED
- Severity: Medium

## Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: PASSED
- Severity: Critical

## Conclusion

In this audit, we thoroughly analyzed JAIHO's Smart Contract. The current code base is well organized but there are promptly some low-level Type issues found in the first phase of Smart Contract Audit, which is acknowledged JAIHO's dev team but as no serious or performance issue with this, they've decided to remain the code unchanged.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

## About eNebula Solutions

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The eNebula Solutions team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

For more information about our security consulting, please mail us at – [contact@enebula.in](mailto:contact@enebula.in)