Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

# SMART CONTRACT

## Security Audit Report

| | |
|---|---|
| Project: | OmniTensor |
| Website: | omnitensor.io |
| Platform: | Ethereum |
| Language: | Solidity |
| Date: | December 10th, 2024 |

# Table of contents






**Email: audit@EtherAuthority.io**

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.


**Email: audit@EtherAuthority.io**

# Introduction

EtherAuthority was contracted by OmniTensor to perform the Security audit of the OmniTensor smart contract code. The audit was performed using manual analysis and automated software tools. This report presents all the findings regarding the audit performed on December 10th, 2024.

**The purpose of this audit was to address the following:**

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

# Project Background

This Solidity code defines a smart contract named `OmniTensor`, which implements the ERC20 token standard with additional functionalities, including owner-based controls, token trading via Uniswap, and tax mechanisms. Here is an overview of its components:

## Key Features:

- **ERC20 Token Basics:**
    - The contract implements the standard ERC20 interface (`IERC20`), providing methods such as `balanceOf`, `transfer`, `approve`, and `transferFrom`.
    - The token is named "OmniTensor" (`OMNIT`) with 18 decimals and a total supply of 1 billion tokens.
- **Ownership Control:** The `Ownable` contract provides functionality for managing ownership, including transferring ownership or renouncing it. The owner has exclusive rights to modify certain contract behavior.
- **Uniswap Integration:**
    - The contract interacts with the Uniswap V2 protocol via `IUniswapV2Factory` and `IUniswapV2Router02` interfaces. It allows adding liquidity in ETH and enables token swaps through the DEX.
    - The `startTrading` function activates trading by creating a Uniswap pair and adding liquidity.
- **Taxation Mechanism:**
    - A buy/sell fee (initially set to 5%) is applied to transactions involving Uniswap. The tax amount is deducted from transfers, except for addresses that are marked as excluded.

    - The contract collects the tax in tokens, which can later be swapped for ETH and distributed to various wallets (`OmegaWallet`, `GammaWallet`, `BetaWallet`, `AlphaWallet`).
- **Limits and Controls:**
    - The contract enforces limits on transaction size (`maxTxValue`) and wallet holdings (`maxWalletHoldings`).
    - Trading can be paused and started by the owner. Additionally, the contract provides the ability to remove transaction and wallet limits.
- **Emergency Functions:**
    - The owner can withdraw accumulated ETH or recover any remaining tokens in the contract.
    - Manual token swaps for ETH can also be triggered by the owner.

This contract is designed with a flexible fee structure, controlled trading features, and strong ownership functionality, making it well-suited for projects aiming to integrate liquidity and token swapping through Uniswap.

# Audit scope

| **Name** | **Code Review and Security Analysis Report for OmniTensor Smart Contract** |
|---|---|
| **Platform** | **Ethereum / Solidity** |
| **File** | OmniTensor.sol |
| **Smart Contract Link** | 0x6f2ee0e1ebfa00015d3040760f6c039f46b6c662 |
| **Audit Date** | December 10th, 2024 |
| **Revised Audit Date** | December 20th, 2024 |

# Claimed Smart Contract Features

| Claimed Feature Detail | Our Observation |
|---|---|
| **Tokenomics:**<br>• Name: OmniTensor<br>• Symbol: OMNIT<br>• Decimals: 18<br>• Total Supply: 1 billion | **YES, This is valid.** |
| **Transaction Limits:**<br>• Maximum transaction value: 1 billion tokens.<br>• Maximum wallet holdings: 1 billion tokens. | **YES, This is valid.** |
| **Tax and Fee Mechanism:**<br>• Buy and sell transactions are subject to a 5% fee by default.<br>• Taxes collected from transactions are converted to ETH, which is distributed to the designated wallets: OmegaWallet, GammaWallet, BetaWallet, and AlphaWallet. | **YES, This is valid.** |

# Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **"Secured"**. The OmniTensor smart contract is 100% decentralized, as it renounces ownership, making it ownerless.


We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

**We found 0 critical, 0 high, 0 medium, 3 low, and 5 very low-level issues.**

**We confirm that all the issues are acknowledged.**

**Investor Advice:** A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.


**Email: audit@EtherAuthority.io**

# Technical Quick Stats

| Main Category | Subcategory | Result |
|---|---|---|
| Contract Programming | The solidity version is not specified | Passed |
| | The solidity version is too old | Passed |
| | Integer overflow/underflow | Passed |
| | Function input parameters lack check | Passed |
| | Function input parameters check bypass | Passed |
| | Function access control lacks management | Passed |
| | Critical operation lacks event log | Moderated |
| | Human/contract checks bypass | Passed |
| | Random number generation/use vulnerability | N/A |
| | Fallback function misuse | Passed |
| | Race condition | Passed |
| | Logical vulnerability | Passed |
| | Features claimed | Passed |
| | Other programming issues | Moderated |
| Code Specification | Function visibility not explicitly declared | Passed |
| | Var. storage location not explicitly declared | Passed |
| | Use keywords/functions to be deprecated | Passed |
| | Unused code | Passed |
| Gas Optimization | “Out of Gas” Issue | Passed |
| | High consumption ‘for/while’ loop | Passed |
| | High consumption ‘storage’ storage | Passed |
| | Assert() misuse | Passed |
| Business Risk | The maximum limit for mintage is not set | Passed |
| | “Short Address” Attack | Passed |
| | “Double Spend” Attack | Passed |

**Overall Audit Result:** **PASSED**


**Email: audit@EtherAuthority.io**

# Business Risk Analysis

| Category | Result |
|---|---|
| Buy Tax | 5% |
| Sell Tax | 5% |
| Cannot Buy | No |
| Cannot Sell | No |
| Max Tax | 5% |
| Modify Tax | Yes |
| Fee Check | No |
| Is Honeypot | Not Detected |
| Trading Cooldown | Not Detected |
| Can Pause Trade? | No |
| Pause Transfer? | Not Detected |
| Max Transaction amount? | Yes |
| Is it Anti-whale? | Not Detected |
| Is Anti-bot? | Not Detected |
| Is it a Blacklist? | Not Detected |
| Blacklist Check | No |
| Can Mint? | No |
| Is it a Proxy? | No |
| Can Take Ownership? | Yes |
| Hidden Owner? | Not Detected |
| Self Destruction? | Not Detected |
| Auditor Confidence | High |

**Overall Audit Result:** **PASSED**


**Email: audit@EtherAuthority.io**

## Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces.  This is a compact and well-written smart contract.

The libraries in the OmniTensor are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the OmniTensor.

The OmniTensor team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

## Documentation

We were given an OmniTensor smart contract code in the form of an etherscan.io weblink.

As mentioned above, the code parts are well commented on. And the logic is straightforward.  So, it is easy to understand the programming flow and complex code logic quickly. Comments are very helpful in understanding the overall architecture of the protocol.

## Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries,  its functions are not used in external smart contract calls.

# AS-IS overview

### Functions

| Sl. | Functions | Type | Observation | Conclusion |
|---|---|---|---|---|
| 1 | constructor | write | Hardcoded addresses | Refer Audit Findings |
| 2 | name | write | Passed | No Issue |
| 3 | symbol | write | Passed | No Issue |
| 4 | decimals | write | Passed | No Issue |
| 5 | totalSupply | write | Passed | No Issue |
| 6 | getFeeRates | external | Passed | No Issue |
| 7 | balanceOf | read | Passed | No Issue |
| 8 | transfer | write | Passed | No Issue |
| 9 | allowance | read | Passed | No Issue |
| 10 | approve | write | Passed | No Issue |
| 11 | transfer from | write | Passed | No Issue |
| 12 | _setAllowance | write | Passed | No Issue |
| 13 | startTrading | external | Centralization risk, Hardcoded addresses, The startTrading needs the Coin and token balance of the token contract | Refer Audit Findings |
| 14 | setExcludedAccount | external | Renounces ownership, making the contract ownerless | No Issue |
| 15 | disableLimits | external | Critical operation lacks an event log, Limits cannot be enabled once disabled | Refer Audit Findings |
| 16 | adjustTaxRates | external | Critical operation lacks event log | Refer Audit Findings |
| 17 | _executeTokenTransfer | write | Passed | No Issue |
| 18 | _executeTransfer | write | Passed | No Issue |
| 19 | withdrawEth | external | Critical operation lacks an event log, The owner can withdraw all the coin and token balance of the contract, Transfer 0 coin | Refer Audit Findings |
| 20 | recoverTokens | external | Critical operation lacks an event log, The owner can withdraw all the coin and token balance of the contract | Refer Audit Findings |
| 21 | executeManualSwap | external | Critical operation lacks event log | Refer Audit Findings |
| 22 | _exchangeTokensForEth | write | Passed | No Issue |

| | | | | |
|---|---|---|---|---|
| **23** | receive | write | Passed | No Issue |
| **24** | owner | read | Passed | No Issue |
| **25** | onlyOwner | modifier | Passed | No Issue |
| **26** | transferOwnership | write | Renounces ownership, making the contract ownerless | No Issue |
| **27** | _updateOwnership | internal | Passed | No Issue |
| **28** | renounceOwnership | write | Renounces ownership, making the contract ownerless | No Issue |

# Severity Definitions

| Risk Level | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc. |
| High | High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial |
| Medium | Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose |
| Low | Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution |
| Lowest / Code Style / Best Practice | Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored. |


**Email: audit@EtherAuthority.io**

# Audit Findings

## Critical Severity

No critical severity vulnerabilities were found.

## High Severity

No high-severity vulnerabilities were found.

## Medium

No medium-severity vulnerabilities were found.

## Low

(1) Centralization risk:

```
/**
     * Starts trading by setting up Uniswap pair and enabling
liquidity.
     */
    function startTrading() external onlyOwner {
        require(!_isTradingActive, "Trading is already enabled");
        _uniswapV2Router =
IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
        uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),
_uniswapV2Router.WETH());
        _setAllowance(address(this), address(_uniswapV2Router),
_totalSupply);

        _uniswapV2Router.addLiquidityETH{value: address(this).balance}(
            address(this),
            balanceOf(address(this)),
            0,
            0,
```

```
            owner(),
            block.timestamp
        );

        IERC20(uniswapV2Pair).approve(address(_uniswapV2Router),
type(uint).max);
        _isTradingActive = true;
        _launchBlock = block.number;
    }
```

When trading starts, added liquidity will be transferred to the Owner's wallet. If the private key of the owner's wallet is compromised, then it will create a problem.

**Resolution:** The owner can accept this risk and handle the private key very securely.
**Status:** Acknowledged

(2) Critical operation lacks event log:
Some events do not have a log which can lead to tracking issues of the variable update.
**Events are:**

- withdrawEth
- recoverTokens
- executeManualSwap
- adjustTaxRates
- disableLimits

**Resolution:** We suggest adding an event log will help to track the methods and some variables' state.
**Status:** Acknowledged

(3) The owner can withdraw all the coin and token balance of the contract:

```
 /**
     * Withdraws ETH from the contract.
     */
    function withdrawEth() external onlyOwner {
        (bool success, ) = owner().call{value:
address(this).balance}("");
        require(success, "Rescue ETH failed");
```

```
    }

    /**
     * Transfers the remaining tokens in the contract to the owner.
     */
    function recoverTokens() external onlyOwner {
        uint256 contractTokenBalance = balanceOf(address(this));
        require(contractTokenBalance > 0, "No tokens to rescue");

        _executeTokenTransfer(address(this), owner(),
contractTokenBalance, 0);
    }
```

The owner can drain all the coin and token balance of the contract. If the private key of the owner's wallet is compromised, then it will create a problem.

**Resolution:** The owner can accept this risk and handle the private key very securely.
**Status:** Acknowledged

## Very Low / Informational / Best practices:

(1) Transfer 0 coin:

```
 /**
     * Withdraws ETH from the contract.
     */
    function withdrawEth() external onlyOwner {
        (bool success, ) = owner().call{value:
address(this).balance}("");
        require(success, "Rescue ETH failed");
    }
```

In the withdrawEth function, there is no check for the contract balance. This can execute a transaction to transfer even for 0 coins which is just a waste of gas.

**Resolution:** We suggest checking the contract balance before transfer to the owner.
**Status:** Acknowledged

(2) Hardcoded addresses:

```
/**
    * Constructor initializes wallets and assigns the total token
supply to the contract deployer.
    */

constructor() {
        OmegaWallet = 0x38106d5664EfAf7aD02E3e5169b1F79591aFc71D; //
must change > Reserve
        GammaWallet = 0xBd92bF5d4f7d1E4E8f3B3E99F99738B9aEEfCC55;
        BetaWallet = 0x38106d5664EfAf7aD02E3e5169b1F79591aFc71D; //
must change > BB
        AlphaWallet = 0x2F6D0A6B2bC5e219eb9a288F63048b911648B9Aa;
```

```
function startTrading() external onlyOwner {
    require(!_isTradingActive, "Trading is already enabled");
    _uniswapV2Router = IUniswapV2Router02(0xD99D1c33F9fC3444f8101754aBC46c52416550D1);
    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this)
    _setAllowance(address(this), address(_uniswapV2Router), _totalSupply);
```

For routers and some wallets, a hardcoded address is used which cannot be changeable.

**Resolution:** We suggest confirming these addresses before deployment.

**Status:** Acknowledged

(3) Variables can be immutable:

```
address OmegaWallet; // Wallet for specific allocation
address GammaWallet;
address BetaWallet;
address AlphaWallet;
```

Variables that are defined within the constructor but further remain unchanged should be marked as immutable to save gas and to ease the reviewing process of third-parties.

**Resolution:** We suggest defining them as Immutable to reduce some gas.

**Status:** Acknowledged

(4) Limits cannot be enabled once disabled:

```
/**
```

```
     * Removes transaction and wallet holding limits.
     */
    function disableLimits() external onlyOwner {
        maxTxValue = _totalSupply;
        maxWalletHoldings = _totalSupply;
    }
```

A limit for maxTxValue and maxWalletHoldings once disabled then no option to enable it.

**Resolution:** We suggest including an option to enable it if necessary.

**Status:** **Acknowledged**

(5) The startTrading needs the Coin and token balance of the token contract:

Before startTrading, the Owner has to transfer tokens and coins to the contract manually.

**Resolution:** We suggest either adding direct transfer in the code or the Owner has to take care of these steps before startTrading

**Status:** **Acknowledged**

## Centralization Risk

The OmniTensor smart contract is **100% decentralized as it renounces ownership, making it ownerless.**

Therefore, there is **no** centralization risk.

# Conclusion

We were given a contract code in the form of an etherscan.io weblink, and we used all possible tests based on the given objects. We have observed 3 low and 5 very low severity issues. We confirm that all smart contract issues are acknowledged. **So, the smart contract is ready for mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all security vulnerabilities and other issues found in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

# Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

**Manual Code Review:**

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

**Vulnerability Analysis:**

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of the functionality of the software under review. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

**Documenting Results:**

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

**Suggested Solutions:**

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

# Disclaimers

## EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

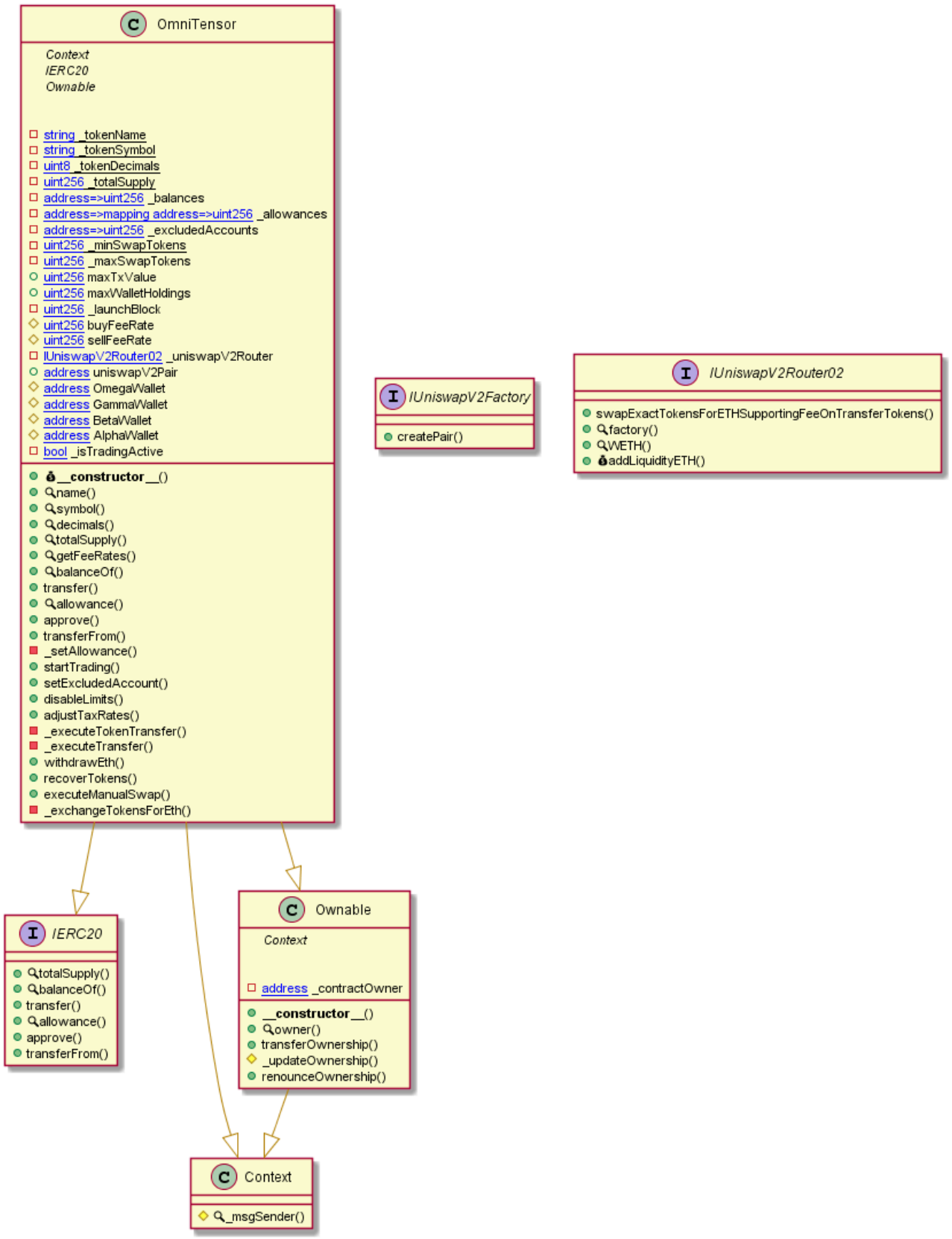
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

## Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

# Appendix

## Code Flow Diagram - OmniTensor



# Slither Results Log

**Slither Log >> OmniTensor.sol**

```
INFO:Detectors:
Reentrancy in OmniTensor._executeTransfer(address,address,uint256)
(OmniTensorensor.sol#265-297):
	External calls sending eth:
	- _exchangeTokensForEth(contractTokenBalance) (OmniTensor.sol#290)
		- (success) = OmegaWallet.call{value: OmegaFund}() (OmniTensor.sol#350)
		- (success,None) = GammaWallet.call{value: GammaFund}() (OmniTensor.sol#351)
		- (success,None) = BetaWallet.call{value: BetaFund}() (OmniTensor.sol#352)
		- (success,None) = AlphaWallet.call{value: AlphaFund}() (OmniTensor.sol#353)
	State variables written after the call(s):
	- _executeTokenTransfer(from,to,amount,taxRate) (OmniTensor.sol#296)
		- _balances[from] -= amount (OmniTensor.sol#255)
		- _balances[to] += transferAmount (OmniTensor.sol#256)
		- _balances[address(this)] += taxAmount (OmniTensor.sol#257)
	OmniTensorensor._balances (OmniTensor.sol#100) can be used in cross function
reentrancies:
	- OmniTensorensor._executeTokenTransfer(address,address,uint256,uint256)
(OmniTensor.sol#251-260)
	- OmniTensorensor.balanceOf(address) (OmniTensor.sol#163-165)
	- OmniTensorensor.constructor() (OmniTensor.sol#126-137)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
OmniTensorensor.startTrading() (OmniTensor.sol#204-222) ignores return value by
_uniswapV2Router.addLiquidityETH{value:
address(this).balance}(address(this),balanceOf(address(this)),0,0,owner(),block.timestamp)
(OmniTensor.sol#210-217)
OmniTensorensor.startTrading() (OmniTensor.sol#204-222) ignores return value by
IERC20(uniswapV2Pair).approve(address(_uniswapV2Router),type()(uint256).max)
(OmniTensor.sol#219)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
OmniTensorensor.adjustTaxRates(uint256,uint256) (OmniTensor.sol#242-246) should emit an
event for:
	- buyFeeRate = newBuyTaxRate (OmniTensor.sol#244)
	- sellFeeRate = newSellTaxRate (OmniTensor.sol#245)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Reentrancy in OmniTensorensor.startTrading() (OmniTensor.sol#204-222):
	External calls:
	- uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WE
```

```
TH()) (OmniTensor.sol#207)
	Event emitted after the call(s):
	- Approval(owner,spender,amount) (OmniTensor.sol#198)
		- _setAllowance(address(this),address(_uniswapV2Router),_totalSupply)
(OmniTensor.sol#208)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Pragma version0.8.20 (OmniTensor.sol#3) necessitates a version too recent to be trusted.
Consider deploying with 0.8.18.
solc-0.8.20 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in OmniTensorensor.withdrawEth() (OmniTensor.sol#302-305):
	- (success) = owner().call{value: address(this).balance}() (OmniTensor.sol#303)
Low level call in OmniTensorensor._exchangeTokensForEth(uint256) (OmniTensor.sol#329-357):
	- (success) = OmegaWallet.call{value: OmegaFund}() (OmniTensor.sol#350)
	- (success,None) = GammaWallet.call{value: GammaFund}() (OmniTensor.sol#351)
	- (success,None) = BetaWallet.call{value: BetaFund}() (OmniTensor.sol#352)
	- (success,None) = AlphaWallet.call{value: AlphaFund}() (OmniTensor.sol#353)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IUniswapV2Router02.WETH() (OmniTensor.sol#82) is not in mixedCase
Constant OmniTensorensor._totalSupply (OmniTensor.sol#98) is not in
UPPER_CASE_WITH_UNDERSCORES
Constant OmniTensorensor._minSwapTokens (OmniTensor.sol#104) is not in
UPPER_CASE_WITH_UNDERSCORES
Variable OmniTensorensor.OmegaWallet (OmniTensor.sol#116) is not in mixedCase
Variable OmniTensorensor.GammaWallet (OmniTensor.sol#117) is not in mixedCase
Variable OmniTensorensor.BetaWallet (OmniTensor.sol#118) is not in mixedCase
Variable OmniTensorensor.AlphaWallet (OmniTensor.sol#119) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
OmniTensorensor.slitherConstructorVariables() (OmniTensor.sol#94-360) uses literals with too
many digits:
	- _maxSwapTokens = 5000000 * 10 ** _tokenDecimals (OmniTensor.sol#105)
OmniTensorensor.slitherConstructorVariables() (OmniTensor.sol#94-360) uses literals with too
many digits:
	- maxTxValue = 5000000 * 10 ** _tokenDecimals (OmniTensor.sol#107)
OmniTensorensor.slitherConstructorVariables() (OmniTensor.sol#94-360) uses literals with too
many digits:
	- maxWalletHoldings = 10000000 * 10 ** _tokenDecimals (OmniTensor.sol#108)
OmniTensorensor.slitherConstructorConstantVariables() (OmniTensor.sol#94-360) uses literals
with too many digits:
	- _totalSupply = 1000000000 * 10 ** _tokenDecimals (OmniTensor.sol#98)
OmniTensorensor.slitherConstructorConstantVariables() (OmniTensor.sol#94-360) uses literals
```

```
with too many digits:
        - _minSwapTokens = 100000 * 10 ** _tokenDecimals (OmniTensor.sol#104)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
OmniTensorensor._maxSwapTokens (OmniTensor.sol#105) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
OmniTensorensor.AlphaWallet (OmniTensor.sol#119) should be immutable
OmniTensorensor.BetaWallet (OmniTensor.sol#118) should be immutable
OmniTensorensor.GammaWallet (OmniTensor.sol#117) should be immutable
OmniTensorensor.OmegaWallet (OmniTensor.sol#116) should be immutable
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:OmniTensor.sol analyzed (6 contracts with 93 detectors), 40 result(s) found
```

# Solidity Static Analysis

## OmniTensor.sol

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
Pos: 341:12:

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
Pos: 353:22:

Gas costs:
Gas requirement of function OmniTensor.withdrawEth is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 302:4:

Gas costs:
Gas requirement of function OmniTensor.executeManualSwap is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 320:4:

Similar variable names:
OmniTensor._executeTransfer(address,address,uint256) : Variables have very similar names "_minSwapTokens" and "_maxSwapTokens". Note: Modifiers are currently not considered by this static analysis.
Pos: 287:45:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 356:8:

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 348:28:

# Solhint Linter

**OmniTensor.sol**

```
Compiler version 0.8.20 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:2
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:29
Error message for require is too long
Pos: 9:53
Function name must be in mixedCase
Pos: 5:81
Contract has 16 states declarations but allowed no more than 15
Pos: 1:93
Constant name must be in capitalized SNAKE_CASE
Pos: 5:103
Variable name must be in mixedCase
Pos: 5:115
Explicitly mark visibility of state
Pos: 5:116
Variable name must be in mixedCase
Pos: 5:116
Explicitly mark visibility of state
Pos: 5:117
Variable name must be in mixedCase
Pos: 5:117
Explicitly mark visibility of state
Pos: 5:118
Variable name must be in mixedCase
Pos: 5:118
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:125
Error message for require is too long
Pos: 13:185
Error message for require is too long
Pos: 17:275
Avoid making time-based decisions in your business logic
Pos: 13:340
Variable name must be in mixedCase
Pos: 9:346
Variable name must be in mixedCase
Pos: 9:347
Code contains empty blocks
Pos: 32:358
```

**Software analysis result:**

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.




**Email: audit@EtherAuthority.io**