# Audit Report
# Generate by X Auditor AI

## Connect With Us

Website: xauditorai.org

Telegram: t.me/xauditoraichannel

# Token Detail

| | |
|---|---|
| **Token Name** | DeepL |
| **Contract Address** | 0xd0bcB2c156a3507670f9BedC319a6409C41bA68E |
| **Token Symbol** | DeepL |
| **Holders** | 100 |
| **Buy Tax** | 5% |
| **Sell Tax** | 5% |
| **is Contract Verified** | Verified |
| **is Proxy Contract** | No |
| **is Honeypot** | No |
| **Anti-Whale Function** | Yes |
| **Mintable Function** | No |
| **Fake Renounce** | No |
| **Hidden Owner** | No |
| **Blacklist Function** | No |
| **Whitelist Function** | Yes |
| **Trading Cooldown Function** | Yes |
| **selfDestruct Function** | No |
| **Transfer Pauseable** | No |
| **Owner Can Change Taxes** | No |
| **Owner Can Change Balance** | No |

# Automated Audit Report

### Solidity assert violation (SWC-110)

**Severity:** *PASSED*

### Integer overflow/underflow (SWC-101)

**Severity:** *PASSED*

### Potential weak source of randomness (SWC-120)

**Severity:** *Low*

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. 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 use of these variables introduces a certain level of trust into miners.

### Uninitialized Storage Variables (SWC-109)

**Severity:** *PASSED*

### Unprotect Withdraw ETH (SWC-105)

**Severity:** *PASSED*

### Loop Over Unbounded Data Structure (SWC-128)

**Severity:** *PASSED*

### Outdated compiler version (SWC-102)

**Severity:** *PASSED*

### Unused State/Local Variable (SWC-131)

**Severity:** *PASSED*

### Deprecated Global Variables/Function (SWC-111)

**Severity:** *PASSED*

### State Variable Visibility (SWC-108)

**Severity:** *PASSED*

# AI Audit Report

### Vulnerabilities Found:

1. **Lack of Proper Access Control**:

- The `transferDelayEnabled`, `swapEnabled`, `tradingOpen`, and `_maxTxAmount` variables can be changed by anyone since there is no access control mechanism in place to restrict these changes to only the contract owner. This can lead to potential misuse of these variables.

2. **Potential Reentrancy Vulnerability**:

- In the `swapTokensForEth` function, there is a call to an external contract (`uniswapV2Router`) to swap tokens for ETH. This external call should be the last action to prevent potential reentrancy vulnerabilities due to an external contract call.

3. **Unsigned Integer Underflow**:

- In the `_balances` subtraction in the `_transfer` function, there is a risk of underflow if the `amount` is greater than the balance of the `from` address. Consider adding a check to ensure that the balance is sufficient before subtracting it.

4. **Front-Running Attack**:

- The current implementation lacks a mechanism to handle potential front-running attacks during token transfers, particularly in the `_transfer` function. Consider implementing safeguards to mitigate the risk of front-running attacks.

### Recommendations:

1. **Access Control**:

- Implement proper access control modifiers like `onlyOwner` to restrict access to critical functions and variables only to the contract owner.

2. **Avoid Reentrancy**:

- Ensure that external contract calls are made as the last action within a function to prevent reentrancy vulnerabilities. Consider using the `nonReentrant` modifier or similar protection methods.

3. **Safe Math Operations**:

- Apply safe math operations using the `SafeMath` library consistently to prevent underflow and overflow errors in integer calculations.

4. **Front-Running Mitigation**:

- Implement techniques like the use of `block.timestamp` or `block.number` to introduce randomness in function execution order to mitigate front-running attacks.

5. **Code Review and Testing**:

- Conduct thorough code reviews and extensive testing to identify and address any potential vulnerabilities or logic flaws in the smart contract.

By addressing these vulnerabilities and implementing the recommended best practices, you can enhance the security and robustness of the smart contract.

# Disclaimer

This audit report is provided for informational purposes only and is not intended to be used as a basis for financial or investment decisions. The findings, interpretations, and conclusions expressed in this Report are based on the information available at the time of the audit and are subject to change without notice.

While every effort has been made to ensure the accuracy and completeness of the analysis, X Auditor AI does not guarantee the correctness, reliability, or completeness of the Report. The smart contract code is subject to inherent risks, including but not limited to coding errors, vulnerabilities, and unforeseen interactions with other smart contracts or blockchain protocols.

X Auditor AI is not liable for any direct, indirect, incidental, special, or consequential damages arising out of the use of this Report. It is the responsibility of the smart contract owner and users to conduct their own due diligence and assess the risks associated with the smart contract.

This Report does not constitute an endorsement or recommendation of the smart contract or its associated project. X Auditor AI does not assume any responsibility for the use or interpretation of the information contained in this Report.

By using this Report, you acknowledge and agree to the terms and conditions outlined in this disclaimer.