Audit Report
Generated 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
Contract Owner	0x0000000000000000000000000000000000000000
Holders	716
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

Ignore some function return Yes if contract renounced and Fake Renounce and/or Hidden Owner is return 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Outdated compiler version (SWC-102)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

Here are some vulnerabilities and potential solutions for the provided smart contract:

1. **Potential Reentrancy Attack:** The `sendETHToFee` function transfers ETH to a specified address. If the destination address is a contract that can call an external function upon receiving ETH, it may lead to a reentrancy attack.

- **Solution:** Use the "Withdrawal Pattern" to separate the transfer of funds and update of state variables. Withdraw the funds first and then update the state variables.

2. **Lack of Access Control:** Certain functions like `manualSwap` should only be accessible to specific addresses to prevent unauthorized calls.

- **Solution:** Implement access control modifiers such as `onlyOwner` for sensitive functions like `manualSwap`.

3. **Front-Running Attacks:** The price impact on swaps could lead to front-running attacks.

- **Solution:** Implement a mechanism to reduce the impact using multi-step functions or other techniques to mitigate front-running.

4. **Integer Overflow:** The SafeMath library already protects against overflow in most arithmetic operations, but double-check everywhere else that arithmetic operations are performed without SafeMath.

- **Solution:** Ensure all arithmetic operations are safe and use SafeMath.

5. **Timestamp Dependency:** The usage of block timestamps for controlling transfer and swap timings might not be secure as it can be manipulated or become inaccurate.

- **Solution:** Consider using a more secure and tamper-resistant approach for time-dependent operations, like block number comparisons or other mechanisms.

6. **Code Duplication:** There are instances of duplicated code in different parts of the contract which can lead to maintenance problems in the future.

- **Solution:** Refactor the code to remove duplication and improve maintainability.

7. **Inefficient Token Transfer:** The `_transfer` function directly transfers tokens, which could be better handled by leveraging the existing `transfer` and `transferFrom` functions.

- **Solution:** Consistently use the ERC20 standard transfer functions for token transfers within the contract.

8. **Public Function to Open Trading:** The `openTrading` function is currently accessible by anyone, which might not be desirable.

- **Solution:** Limit access to the `openTrading` function to only authorized addresses, such as the contract owner.

Remember to thoroughly test any changes to ensure they do not introduce new vulnerabilities. It's recommended to seek a professional audit by experts in Solidity and smart contract security for a comprehensive evaluation of the contract's security posture.

Disclaimer