Audit Report for **Treasury**

Date: 10 January 2025

Audit result: **Passed with Medium**

**Token Address:** -

**Name:** Treasury

**Symbol:** -

**Decimals**: 18

**Network:** Ether Scan

**Token Type**: ERC-20

**Owner**: 0x26aBD5b55B8b02FcC3037A1B0B0E9E99C19106c8

**Deployer:** 0x26aBD5b55B8b02FcC3037A1B0B0E9E99C19106c8

**Token Supply: -**

**Checksum:** b45f4a3f449246571a8b891ae4723d05

**Token Overview:**

**Buy Fee:** 0-0%

**Sell Fee:** 0-0%

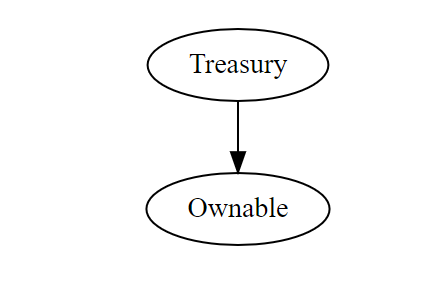
**Transfer Fee:** 0-0%

**Fee Privilege:** Owner

**Ownership:** Owned

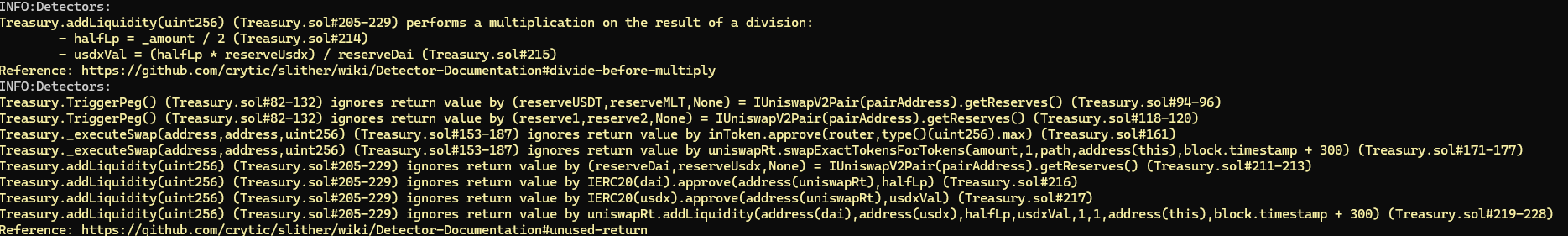
**Minting:** None

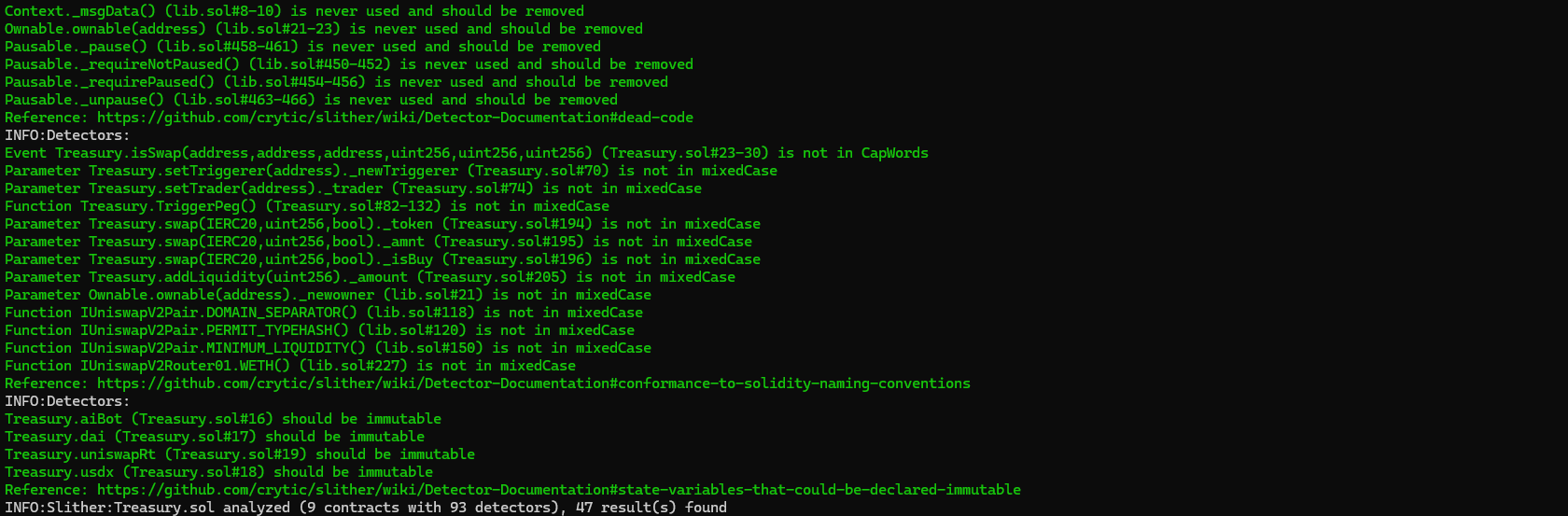
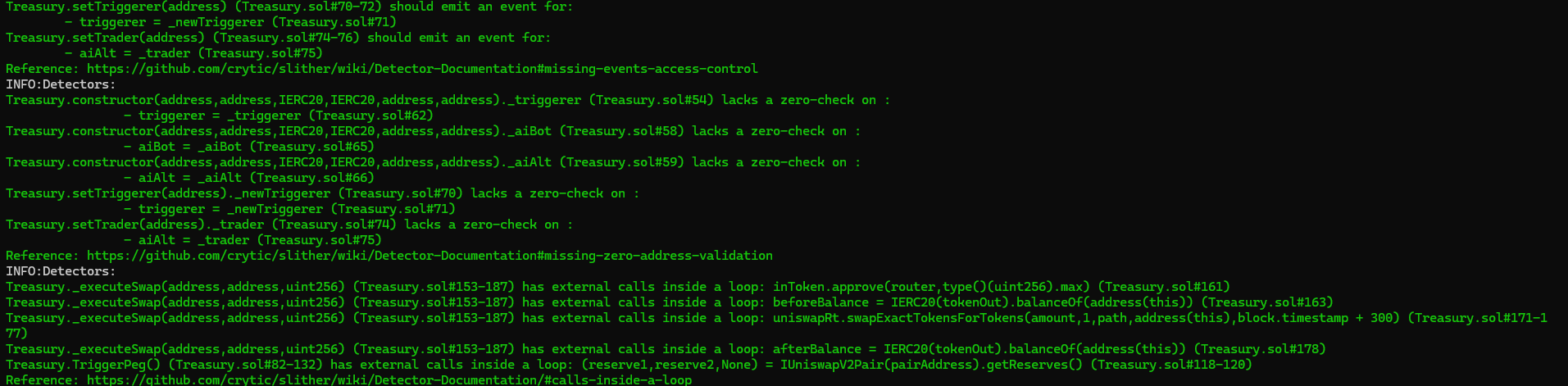
**Max Tx:** No  
**Blacklist:** No  
  
  
  
  
  
  
  
  
  
**Inheritance Tree**

****

**StaticAnalysis**

A static analysis of the code was performed using Slither. No issues were found.





**Ownership Privileges:**

- The owner can setTriggerer.

- The owner can setTrader.

- The owner can setMaintainPriceOnOff.

- The owner can open trading..

- The owner can add/delete bots.

- The owner can manual swap.  
- The owner can rescueEth and rescue Nowar.

**Findings:**

**Critical**: 0

**High**: 2

**Medium**: 3

**Low**: 4

**Informational** **&** **Optimizations**: 1

**Centralization – Enabling Trades**

**Severity**: **High**

**Function**: openTrading

**Status:** Open

**Overview:**

The **EnableTrading** function permits only the contract owner to activate trading capabilities. Until this function is executed, no investors can buy, sell, or transfer their tokens. This places a high degree of control and centralization in the hands of the contract owner.

*function* openTrading() external {

        require(hasRole(*ADMIN\_ROLE*, **msg.sender**), "NOWAR: Not an admin");

        require(!tradingOpen, "NOWAR: Trading is already open");

        swapEnabled **=** true;

        tradingOpen **=** true;

**Suggestion:**

To reduce centralization and potential manipulation, consider one of the following approaches:

1. Automatically enable trading after a specified condition, such as the completion of a presale, is met.
2. If manual activation is still desired, consider transferring the ownership of the contract to a trustworthy, third-party entity like a certified "PinkSale Safu" developer. This can give investors more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad-faith actions by the original owner.

**Centralization – Owner can blacklist wallets.**

**Severity**: **High**

**Function**: addbot, delBots

**Status:** Open

**Overview:**

The owner can blacklist wallets from transferring of tokens for an indefinite period of time which is not recommended. Which can lock user’s token.

*function* addBot(address botAddress\_) public {

        require(hasRole(*ADMIN\_ROLE*, **msg.sender**), "NOWAR: Not a admin");

        require(botAddress\_ !**=** address(*0*), "NOWAR: Invalid address");

        bots[botAddress\_] **=** true;

**emit** BotAdded(botAddress\_);

    }

*function* delBots(address nobotAddress\_) public {

        require(hasRole(*ADMIN\_ROLE*, **msg.sender**), "NOWAR: Not a admin");

        require(nobotAddress\_ !**=** address(*0*), "NOWAR: Invalid address");

        bots[nobotAddress\_] **=** false;

**emit** BotRemoved(nobotAddress\_);

    }

**Suggestion:** There should be a locking period so that the wallet cannot be locked for an indefinite. Period of time.  
 **Centralization – The owner can regain ownership.**

**Severity**: **Medium**

**Function**: pendingOwner

**Status:** Open

**Overview:**

The owner can regain ownership after transferring it with the following steps:

1. Call lock function to set previous owner to the own address

2. Call unlock function to get ownership back

3. Transfer/renounce ownership

4. Call unlock function to get ownership back

abstract *contract* Ownable2Step *is* Context, AccessControl {

    address private \_owner;

    address private \_newOwnerCandidate;

    uint private changeOwnershipTimestamp;

    uint256 private constant onwerDelayTime = *7* days;

*event* OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

*event* NewOwnerCandidateSet(address indexed previousCandidate, address indexed newCandidate);

*event* OwnershipRecoveryInitiated(address indexed currentOwner, address indexed newOwnerCandidate);

*event* OwnershipRecovered(address indexed previousOwner, address indexed newOwner);

**constructor** () {

        address msgSender = \_msgSender();

        require(msgSender != address(*0*), "Invalid message sender address");

        \_owner = msgSender;

**emit** OwnershipTransferred(address(*0*), msgSender);

    }

*function* owner() public view returns (address) {

        return \_owner;

    }

*function* newOwnerCandidate() public view returns (address) {

        return \_newOwnerCandidate;

    }

*function* transferOwnership(address newOwnerCandidate\_) public {

        require(**msg.sender** == \_owner, "Caller not onwer");

        require(newOwnerCandidate\_ != address(*0*), "Invalid new owner candidate address");

        require(newOwnerCandidate\_ != \_owner, "New onwer is the same as the current one");

        \_newOwnerCandidate = newOwnerCandidate\_;

        changeOwnershipTimestamp = **block**.timestamp + onwerDelayTime; *// 7 days delay for safety*

**emit** NewOwnerCandidateSet(\_newOwnerCandidate, newOwnerCandidate\_);

    }

*function* acceptOwnership() public {

        require(\_newOwnerCandidate != address(*0*), "Invalid new owner candidate address");

        require(**msg.sender** == \_newOwnerCandidate, "Only new owner candidate can initiate recovery");

        require(**block**.timestamp >= changeOwnershipTimestamp, "Ownership change period not elapsed");

        address previousOwner = \_owner;

        \_owner = \_newOwnerCandidate;

**delete** \_newOwnerCandidate;

**emit** OwnershipRecovered(previousOwner, \_owner);

**emit** OwnershipTransferred(previousOwner, \_owner);

        \_revokeRole(*ADMIN\_ROLE*, previousOwner);

        \_grantRole(*ADMIN\_ROLE*, \_owner);

    }

*function* cancelOwnershipRecovery() public {

        require(**msg.sender** == \_owner, "Caller not onwer");

        require(\_newOwnerCandidate != address(*0*), "No ownership recovery in progress");

**delete** \_newOwnerCandidate;

**emit** NewOwnerCandidateSet(\_newOwnerCandidate, address(*0*));

    }

}

**Suggestion:** Make sure to set the previous ownership back to address zero after using the unlock function.

**Centralization – Missing Require Check.**

**Severity**: **Medium**

**Function**: setFeeWallet

**Status:** Open

**Overview:**

The owner can set any arbitrary address excluding zero address as this is not recommended because if the owner will set the address to the contract address, then the Eth will not be sent to that address and the transaction will fail and this will lead to a potential honeypot in the contract.

*function* setFeeWallet(address newFeeWallet\_) public {

        require(hasRole(*ADMIN\_ROLE*, **msg.sender**), "NOWAR: Not an admin");

        require(newFeeWallet\_ !**=** address(*0*), "Invalid address: zero address");

        require(newFeeWallet\_ !**=** \_feeWallet, "New fee wallet address is the same as the current one");

        address payable previousFeeWallet\_ **=** \_feeWallet;

        \_isExcludedFromFee[previousFeeWallet\_] **=** false;

        \_isExcludedFromFee[newFeeWallet\_] **=** true;

        \_feeWallet **=** payable(newFeeWallet\_);

**emit** FeeWalletUpdated(previousFeeWallet\_, newFeeWallet\_);

    }

**Suggestion:** It is recommended that the address should not be able to set as a contract address.  
  
**Centralization – Liquidity is added to EOA.**

**Severity**: **Medium**

**function**: addLiquidityETH

**Status:** Open

**Overview:**

Liquidity is added to EOA. It may be drained by the owner.

 IUniswapV2Router02 \_uniswapV2Router **=** IUniswapV2Router02(0xD99D1c33F9fC3444f8101754aBC46c52416550D1);

        uniswapV2Router **=** \_uniswapV2Router;

        \_approve(address(**this**), address(uniswapV2Router), \_tTotal);

        uniswapV2Pair **=** IUniswapV2Factory(\_uniswapV2Router.factory()).createPair(address(**this**), \_uniswapV2Router.*WETH*());

        uniswapV2Router.addLiquidityETH{value**:** address(**this**).balance}(

            address(**this**),

            balanceOf(address(**this**)),

*0*,

*0*,

            owner(),

**block**.timestamp

        )

**Suggestion:**

It is suggested that the address should be a contract address or a dead address.

**Centralization – Missing Events**

**Severity**: **Low**

**Subject**: Missing Events

**Status:** Open

**Overview:**

They serve as a mechanism for emitting and recording data onto the blockchain, making it transparent and easily accessible.

*function* getFeeWallet() public view returns (address payable) {

    }

*function* addAdmin(address newAdmin) public {

        require(hasRole(*ADMIN\_ROLE*, **msg.sender**), "NOWAR: not an admin");

        \_grantRole(*ADMIN\_ROLE*, newAdmin);

    }

*function* removeAdmin(address delAdmin) public {

        require(hasRole(*ADMIN\_ROLE*, **msg.sender**), "NOWAR: not an admin");

        require(delAdmin != owner(), "NOWAR: can not remove onwer");

        \_revokeRole(*ADMIN\_ROLE*, delAdmin);

    }

**Suggestion:** Emit an event for critical changes.

**Centralization – Local variable Shadowing**

**Severity**: **Low**

**Subject**: Variable Shadowing

**Status:** Open

**Overview:**

*function* \_approve(address owner, address spender, uint256 amount) private {

        require(owner != address(*0*), "NOWAR: approve from the zero address");

        require(spender != address(*0*), "NOWAR: approve to the zero address");

        \_allowances[owner][spender] = amount;

**emit** Approval(owner, spender, amount);

    }

*function* allowance(address owner, address spender) public view override returns (uint256) {

        return \_allowances[owner][spender];

    }

**Suggestion:**

Rename the local variables that shadow another component.  
 **Centralization – Missing Zero Address**

**Severity**: **Low**

**Subject**: Zero Check

**Status:** Open

**Overview:**

functions can take a zero address as a parameter (0x00000...). If a function parameter of address type is not properly validated by checking for zero addresses, there could be serious consequences for the contract's functionality.

*function* getFeeWallet() public view returns (address payable) {

    }

**Suggestion:**

It is suggested that the address should not be zero or dead.

**Centralization – Remove the safe math library.**

**Severity**: **Low**

**Status:** Open

**Line Number: 11-46**

**Overview:**

The Safe Math library is no longer needed for Solidity version 0.8 and above. This is because Solidity 0.8 includes checked arithmetic operations by default. All of Safe Math’s methods are now inherited into Solidity programming.

**Optimization**

**Severity**: **Informational**

**Subject**: Floating Pragma.

**Status:** Open

**Overview:**

It is considered best practice to pick one compiler version and stick with it. With a floating pragma, contracts may accidentally be deployed using an outdated.

***pragma*** solidity ^0.8.0;

**Suggestion:**

Adding the latest constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.