BLOCKHAT SECURITY

MindX

Smart Contract Security Audit

Prepared by BlockHat

April 10th, 2024 - April 14th, 2024

BlockHat.io

contact@blockhat.io

Document Properties

Client	mindx
Version	1.0
Classification	Private

Scope

The MindX Contract in the MindX Repository

Link	Address
-	-

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

mindx engaged BlockHat to conduct a security assessment on the MindX beginning on April 10th, 2024 and ending April 14th, 2024. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About MindX

Issuer	mindx
Website	`https://mindx.bot/`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
			Likelihood	

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the MindX implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 1 critical-severity, 3 high-severity, 3 medium-severity, 2 low-severity, 4 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
Potential Locking of ETH in Contract	CRITICAL	Not fixed
Bot Detection Mechanism	HIGH	Not fixed
Taxation Calculation on Bot Transactions	HIGH	Not fixed
Incomplete Bot Detection Condition	HIGH	Not fixed
Centralization of Critical Functional Controls	MEDIUM	Not fixed
Function Naming Clarity for enableTrading	MEDIUM	Not fixed
Error Handling in taxChange Function	MEDIUM	Not fixed
Optimizing Share Distribution Calculations	LOW	Not fixed
Mint vs Transfer for Initial Distribution	LOW	Not fixed
Improving Code Readability in Token Minting	INFORMATIONAL	Not fixed
Utilization of Total Supply Variable	INFORMATIONAL	Not fixed
Naming Clarity	INFORMATIONAL	Not fixed
Redundant Timestamp Assignment	INFORMATIONAL	Not fixed

3 Finding Details

A MindX.sol

A.1 Potential Locking of ETH in Contract [CRITICAL]

Description:

The concern is that some amount of ETH could remain in the contract after the execution of swapAndReward, potentially due to rounding errors or inefficiencies in the swap function.

Listing 1: Mindx.sol

```
    function swapAndReward(uint256 contractTokenBalance) private lockTheSwap {
        // split the contract balance into halves
        uint256 half = contractTokenBalance / 2;

        uint256 otherHalf = contractTokenBalance - half;

        uint256 initialBalance = address(this).balance;

        swapTokensForEth(half);

        uint256 newBalance = address(this).balance - initialBalance;

        // Send ETH amount of newBalance to TreasuryOwner
        payable(TreasuryOwner).transfer(newBalance);
        uint _owner_share = (otherHalf / 100) * OwnerShare;
        uint _revenue_share = otherHalf - _owner_share;
        super._transfer(address(this), TreasuryRevenue, _revenue_share);
        super._transfer(address(this), TreasuryOwner, _owner_share);

    }
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

We recommend adding a withdraw BNB function.

Status - Not fixed

A.2 Bot Detection Mechanism [HIGH]

Description:

The bot detection mechanism flags addresses based solely on the transaction being in the same block, potentially leading to false positives. Additionally, the mechanism only handles bot marking on the buying side, ignoring the selling cases.

Listing 2: Mindx.sol

```
            if( isBot[from] ) {
                super._transfer(from, TreasuryOwner, (amount / 100) *
                    ↪ initial_tax);
            }
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

Redesign the bot detection logic to include both buying and selling activities. Introduce a more robust set of criteria for bot detection to reduce false positives, potentially including a combination of transaction frequency, amount, and behavior over time.

Status - Not fixed

A.3 Taxation Calculation on Bot Transactions [HIGH]

Description:

If an address is flagged as a bot, the function still proceeds to calculate and deduct taxes after already transferring an initial tax, potentially resulting in incorrect final transfer amounts.

Listing 3: Mindx.sol

```
        super._transfer(from, to, amount - Taxation);
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Ensure that once an address is flagged as a bot and penalized, further taxation calculations are adjusted accordingly or halted to prevent double taxation or incorrect token transfers.

Status - Not fixed

A.4 Incomplete Bot Detection Condition [HIGH]

Description:

The bot detection mechanism in the _transfer function potentially fails to accurately identify bots when from is a pair address. This specific condition appears to be unhandled, leading to possible scenarios where bots could manipulate the system by avoiding detection.

Listing 4: Mindx.sol

```
                if(_isExcludedMaxTransactionAmount[to]){
                    isBot[from] = true;
                }
```

```
                    isBot[to] = true;
                }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Implement comprehensive bot detection logic that includes checks for both from and to addresses in relation to pair addresses.

Status - Not fixed

A.5 Centralization of Critical Functional Controls [MEDIUM]

Description:

The Mindx smart contract grants the owner unilateral control over several critical functions including enabling/disabling trading, managing automated market maker pairs, exclusion lists, and token burning. This concentration of power poses significant risks:

- Trading Control: The owner can toggle trading, potentially manipulating the market.
- AMM and Exclusion List Management: The owner can selectively manage which addresses are exempt from transaction rules or are considered automated market makers, potentially leading to unfair advantages and liquidity issues.
- Token Burning: The owner's ability to burn tokens at will can unpredictably affect the token supply and its market dynamics.

Listing 5: Mindx.sol

```
    function enableTrading(bool _status) external onlyOwner {

        tradingEnabled = _status;
        emit enable_trading(_status);
    }
```

Listing 6: Mindx.sol

```
    function adding_isExcludedMaxTransactionAmount(address _a) public
       ↪ onlyOwner{
        _isExcludedMaxTransactionAmount[_a] = true;
        initial_inject_timestamp = block.timestamp;
        tradingEnabled = true;

        emit adding_isExcluded(_a);
    }

    function removing_isExcludedMaxTransactionAmount(address _a) public
       ↪ onlyOwner{
        delete _isExcludedMaxTransactionAmount[_a];
        emit removing_isExcluded(_a);
    }

    function adding_automatedMarketMakerPairs(address _a) public
       ↪ onlyOwner {
        _automatedMarketMaker[_a] = true;
        emit adding_automated(_a);
    }

    function removing_automatedMarketMakerPairs(address _a) public
       ↪ onlyOwner{
        delete _automatedMarketMaker[_a];
        emit removing_automated(_a);
    }
```

Listing 7: Mindx.sol

```
    function BurnDevToken(uint256 amount) public onlyOwner {
        _burn(owner(), amount);
    }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

- Decentralized Governance Implementation: Transition these controls to a decentralized governance framework, such as through a DAO or a committee system using multisig wallets.
- Introduction of Timelocks: Implement timelocks for critical actions like toggling trading or modifying smart contract parameters.
- Transparent and Rule-Based Criteria: Develop clear, documented criteria for managing automated market maker pairs and exclusion lists. This helps ensure that changes are justified, transparent, and not subject to misuse.

Status – Not fixed

A.6 Function Naming Clarity for enableTrading [MEDIUM]

Description:

The function enableTrading toggles the trading status but is named as if it only enables trading, which could mislead users or developers regarding its functionality.

Listing 8: Mindx.sol

```
    function enableTrading(bool _status) external onlyOwner {

        tradingEnabled = _status;
        emit enable_trading(_status);
    }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Rename the function to toggleTrading to more accurately reflect that it can both enable and disable trading. This change improves clarity and reduces the potential for misuse or confusion.

Status - Not fixed

A.7 Error Handling in taxChange Function [MEDIUM]

Description:

The taxChange function uses generic error messages which do not specify the exact cause of the revert, reducing the function's usability and debuggability.

Listing 9: Mindx.sol

```
    function taxChange(uint _b, uint _s) external onlyOwner {
        if(_b > 20){
            revert("The wrong number inputed");
        }
        if(_b < 0){
           revert("The wrong number inputed");
        }
        if(_s > 20){
            revert("The wrong number inputed");
        }
        if(_s < 0){
           revert("The wrong number inputed");
        }
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Use specific error messages for each validation check. For instance, differentiate between errors arising from the buy tax being too high, too low, or the sell tax conditions. This can be done by using messages like ”Buy tax must be between 0 and 20” and ”Sell tax must be between 0 and 20”.

Status - Not fixed

A.8 Optimizing Share Distribution Calculations [LOW]

Description:

Shares for different stakeholders (e.g., TechTeam, Marketing) are calculated and then transferred in a way that might use extra gas due to the creation of intermediate variables.

Listing 10: Mindx.sol

```
        _mint(owner(), 240 * 1e24); //240M
        uint total_Supply = balanceOf(owner());

        uint techTeam_share = (total_Supply / 100) * 5;
        transfer(TechTeam, techTeam_share);

        uint Marketing_share = (total_Supply / 100) * 15;
        transfer(Marketing, Marketing_share);

        uint CEX_share = (total_Supply / 100) * 10;
        transfer(CEX,CEX_share);

        transfer(PreSale,63_370_000_000_000_000_000_000_000);
        transfer(Surplus,28_130_000_000_000_000_000_000_000);

        uint CReward_share = (total_Supply / 100) * 10;
        transfer(CReward,CReward_share);
```

Risk Level:

Likelihood – 3

Impact - 2

Recommendation:

Directly compute and transfer shares in the transfer function call to reduce gas usage, e.g., transfer(TechTeam, (totalSupply() / 100) * 5).

Status - Not fixed

A.9 Mint vs Transfer for Initial Distribution [LOW]

Description:

The use of the transfer function for distributing shares to various stakeholders raises questions about whether the shares are being deducted from the owner or additional minting should be considered.

Listing 11: Mindx.sol

```
        _mint(owner(), 240 * 1e24); //240M
        uint total_Supply = balanceOf(owner());

        uint techTeam_share = (total_Supply / 100) * 5;
        transfer(TechTeam, techTeam_share);

        uint Marketing_share = (total_Supply / 100) * 15;
        transfer(Marketing, Marketing_share);
        uint CEX_share = (total_Supply / 100) * 10;
        transfer(CEX,CEX_share);

        transfer(PreSale,63_370_000_000_000_000_000_000_000);

        transfer(Surplus,28_130_000_000_000_000_000_000_000);

        uint CReward_share = (total_Supply / 100) * 10;
        transfer(CReward,CReward_share);
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Consider using the mint function for initial distributions to clearly define token allocations without impacting the owner's holdings, ensuring transparency and accuracy in token distribution.

Status - Not fixed

A.10 Improving Code Readability in Token Minting [INFORMATIONAL]

Description:

The minting of tokens uses 240 * 1e24, which could be simplified for better readability and to avoid potential errors during code modifications.

Listing 12: Mindx.sol

```
        _mint(owner(), 240 * 1e24); //240M
```

Risk Level:

Likelihood – 2
Impact - 1

Recommendation:

It's recommended to use 240e24 to express the minting amount, enhancing the clarity and reducing potential misunderstandings.

Status - Not fixed

A.11 Utilization of Total Supply Variable [INFORMATIONAL]

Description:

The total_Supply is redundantly calculated by getting the owner's balance post-minting, which is unnecessary since totalSupply() function provides this value inherently.

Listing 13: Mindx.sol

```
        uint total_Supply = balanceOf(owner());
```

Risk Level:

Likelihood – 3
Impact - 2

Recommendation:

Replace balanceOf(owner()) with totalSupply() directly after minting to optimize gas costs and rely on ERC20's built-in functionality.

Status - Not fixed

A.12 Naming Clarity [INFORMATIONAL]

Description:

The mapping _isExcludedMaxTransactionAmount is used to determine if an address is excluded from maximum transaction checks, but its naming suggests broader or different functionality.

Listing 14: Mindx.sol

```
    mapping(address => bool) public _isExcludedMaxTransactionAmount;
```

Risk Level:

Likelihood – 3
Impact - 2

Recommendation:

Rename _isExcludedMaxTransactionAmount to more accurately reflect its purpose, such as _isPairAddress, to avoid confusion and improve code readability.

Status - Not fixed

A.13 Redundant Timestamp Assignment [INFORMATIONAL]

Description:

The function assigns timestamps to _tierTimestamp mapping for both from and to addresses in every transaction without clear purpose or usage documented in the function.

Listing 15: Mindx.sol

```
        _tierTimestamp[to] = block.timestamp;
        _tierTimestamp[from] = block.timestamp;
```

Risk Level:

Likelihood – 2
Impact - 1

Recommendation:

Clarify the purpose of _tierTimestamp in the code documentation or remove this functionality if it is unused to save gas and reduce contract complexity.

Status – Not fixed

4 Best Practices

BP.1 Improving Numeric Constants Representation

Description:

Numeric constants for initial distributions to PreSale and Surplus are not using power notation which affects readability.

Code:

Listing 16: Mindx.sol

```
        transfer(PreSale,63_370_000_000_000_000_000_000_000);

        transfer(Surplus,28_130_000_000_000_000_000_000_000);
```

BP.2 Unnecessary Initialization of Taxation Variable

Description:

The variable Taxation is initialized to zero at the start of the function, which is redundant since it is conditionally set later.

Code:

Listing 17: Mindx.sol

```
        uint Taxation = 0;
        if (_automatedMarketMaker[from] || _automatedMarketMaker[to]) {
            Taxation = 0;
```

5 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
Reentrancy in Mindx._transfer(address,address,uint256) (MindX.sol
   ↪ #815-880):
       External calls:
       - swapAndReward(contractTokenBalance) (MindX.sol#833)
               - uniswapV2Router.swapExactTokensForETH(tokenAmount,0,path
                   ↪ ,address(this),block.timestamp) (MindX.sol#911-917)
       External calls sending eth:
       - swapAndReward(contractTokenBalance) (MindX.sol#833)
               - address(TreasuryOwner).transfer(newBalance) (MindX.sol
                   ↪ #895)
       State variables written after the call(s):
       - super._transfer(from,TreasuryOwner,(amount / 100) * initial_tax
           ↪ ) (MindX.sol#865)
               - _balances[sender] = senderBalance - amount (MindX.sol
                   ↪ #616)
               - _balances[recipient] += amount (MindX.sol#618)
       ERC20._balances (MindX.sol#492) can be used in cross function
           ↪ reentrancies:
       - ERC20._burn(address,uint256) (MindX.sol#637-652)
       - ERC20._mint(address,uint256) (MindX.sol#625-635)
       - ERC20._transfer(address,address,uint256) (MindX.sol#600-623)
```

```
        - ERC20.balanceOf(address) (MindX.sol#522-526)
        - super._transfer(from,address(this),Taxation) (MindX.sol#873)
               - _balances[sender] = senderBalance - amount (MindX.sol
                  ↪ #616)
               - _balances[recipient] += amount (MindX.sol#618)
        ERC20._balances (MindX.sol#492) can be used in cross function
           ↪ reentrancies:
        - ERC20._burn(address,uint256) (MindX.sol#637-652)
        - ERC20._mint(address,uint256) (MindX.sol#625-635)
        - ERC20._transfer(address,address,uint256) (MindX.sol#600-623)
        - ERC20.balanceOf(address) (MindX.sol#522-526)
        - super._transfer(from,to,amount - Taxation) (MindX.sol#879)
               - _balances[sender] = senderBalance - amount (MindX.sol
                  ↪ #616)
               - _balances[recipient] += amount (MindX.sol#618)
        ERC20._balances (MindX.sol#492) can be used in cross function
           ↪ reentrancies:
        - ERC20._burn(address,uint256) (MindX.sol#637-652)
        - ERC20._mint(address,uint256) (MindX.sol#625-635)
        - ERC20._transfer(address,address,uint256) (MindX.sol#600-623)
        - ERC20.balanceOf(address) (MindX.sol#522-526)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities
INFO:Detectors:
Mindx.constructor(address) (MindX.sol#733-771) performs a multiplication
   ↪  on the result of a division:
        - techTeam_share = (total_Supply / 100) * 5 (MindX.sol#751)
Mindx.constructor(address) (MindX.sol#733-771) performs a multiplication
   ↪  on the result of a division:
        - Marketing_share = (total_Supply / 100) * 15 (MindX.sol#754)
Mindx.constructor(address) (MindX.sol#733-771) performs a multiplication
   ↪  on the result of a division:
        - CEX_share = (total_Supply / 100) * 10 (MindX.sol#757)
```

```
Mindx.constructor(address) (MindX.sol#733-771) performs a multiplication
    ↪  on the result of a division:
        - CReward_share = (total_Supply / 100) * 10 (MindX.sol#764)
Mindx._transfer(address,address,uint256) (MindX.sol#815-880) performs a
    ↪ multiplication on the result of a division:
        - Taxation = (amount / 100) * Taxation (MindX.sol#870)
Mindx._transfer(address,address,uint256) (MindX.sol#815-880) performs a
    ↪ multiplication on the result of a division:
        - super._transfer(from,TreasuryOwner,(amount / 100) * initial_tax
            ↪ ) (MindX.sol#865)
Mindx.swapAndReward(uint256) (MindX.sol#882-901) performs a
    ↪ multiplication on the result of a division:
        - _owner_share = (otherHalf / 100) * OwnerShare (MindX.sol#896)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #divide-before-multiply
INFO:Detectors:
Mindx._transfer(address,address,uint256) (MindX.sol#815-880) uses a
    ↪ dangerous strict equality:
        - _transactorLastblock[tx.origin] == block.number (MindX.sol#855)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dangerous-strict-equalities
INFO:Detectors:
Mindx._transfer(address,address,uint256) (MindX.sol#815-880) uses tx.
    ↪ origin for authorization: _transactorLastblock[tx.origin] ==
    ↪ block.number (MindX.sol#855)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dangerous-usage-of-txorigin
INFO:Detectors:
Mindx.taxChange(uint256,uint256) (MindX.sol#781-799) contains a
    ↪ tautology or contradiction:
        - _b < 0 (MindX.sol#785)
Mindx.taxChange(uint256,uint256) (MindX.sol#781-799) contains a
    ↪ tautology or contradiction:
        - _s < 0 (MindX.sol#791)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #tautology-or-contradiction
INFO:Detectors:
Mindx.swapTokensForEth(uint256) (MindX.sol#903-918) ignores return value
   ↪  by uniswapV2Router.swapExactTokensForETH(tokenAmount,0,path,
   ↪ address(this),block.timestamp) (MindX.sol#911-917)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #unused-return
INFO:Detectors:
Reentrancy in Mindx._transfer(address,address,uint256) (MindX.sol
   ↪ #815-880):
        External calls:
        - swapAndReward(contractTokenBalance) (MindX.sol#833)
                - uniswapV2Router.swapExactTokensForETH(tokenAmount,0,path
                   ↪ ,address(this),block.timestamp) (MindX.sol#911-917)
        External calls sending eth:
        - swapAndReward(contractTokenBalance) (MindX.sol#833)
                - address(TreasuryOwner).transfer(newBalance) (MindX.sol
                   ↪ #895)
        State variables written after the call(s):
        - _tierTimestamp[to] = block.timestamp (MindX.sol#876)
        - _tierTimestamp[from] = block.timestamp (MindX.sol#877)
        - _transactorLastblock[tx.origin] = block.number (MindX.sol#867)
        - isBot[from] = true (MindX.sol#858)
        - isBot[to] = true (MindX.sol#860)
Reentrancy in Mindx.swapAndReward(uint256) (MindX.sol#882-901):
        External calls:
        - swapTokensForEth(half) (MindX.sol#890)
                - uniswapV2Router.swapExactTokensForETH(tokenAmount,0,path
                   ↪ ,address(this),block.timestamp) (MindX.sol#911-917)
        External calls sending eth:
        - address(TreasuryOwner).transfer(newBalance) (MindX.sol#895)
        State variables written after the call(s):
```

```
       - super._transfer(address(this),TreasuryRevenue,_revenue_share) (
          ↪ MindX.sol#898)
              - _balances[sender] = senderBalance - amount (MindX.sol
                 ↪ #616)
              - _balances[recipient] += amount (MindX.sol#618)
       - super._transfer(address(this),TreasuryOwner,_owner_share) (
          ↪ MindX.sol#899)
              - _balances[sender] = senderBalance - amount (MindX.sol
                 ↪ #616)
              - _balances[recipient] += amount (MindX.sol#618)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Mindx._transfer(address,address,uint256) (MindX.sol
   ↪ #815-880):
       External calls:
       - swapAndReward(contractTokenBalance) (MindX.sol#833)
              - uniswapV2Router.swapExactTokensForETH(tokenAmount,0,path
                 ↪ ,address(this),block.timestamp) (MindX.sol#911-917)
       External calls sending eth:
       - swapAndReward(contractTokenBalance) (MindX.sol#833)
              - address(TreasuryOwner).transfer(newBalance) (MindX.sol
                 ↪ #895)
       Event emitted after the call(s):
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(from,to,amount - Taxation) (MindX.sol
                 ↪ #879)
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(from,address(this),Taxation) (MindX.sol
                 ↪ #873)
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(from,TreasuryOwner,(amount / 100) *
                 ↪ initial_tax) (MindX.sol#865)
Reentrancy in Mindx.swapAndReward(uint256) (MindX.sol#882-901):
       External calls:
       - swapTokensForEth(half) (MindX.sol#890)
              - uniswapV2Router.swapExactTokensForETH(tokenAmount,0,path
                 ↪ ,address(this),block.timestamp) (MindX.sol#911-917)
       External calls sending eth:
       - address(TreasuryOwner).transfer(newBalance) (MindX.sol#895)
       Event emitted after the call(s):
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(address(this),TreasuryOwner,_owner_share
                 ↪ ) (MindX.sol#899)
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(address(this),TreasuryRevenue,
                 ↪ _revenue_share) (MindX.sol#898)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-3
INFO:Detectors:
Mindx._transfer(address,address,uint256) (MindX.sol#815-880) uses
   ↪ timestamp for comparisons
       Dangerous comparisons:
       - require(bool,string)(initial_inject_timestamp > 0,not paired
          ↪ yet) (MindX.sol#843)
       - require(bool,string)(block.timestamp - initial_inject_timestamp
          ↪  > second_buy_limit,not enabled yet) (MindX.sol#844)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #block-timestamp
INFO:Detectors:
Address._revert(bytes,string) (MindX.sol#474-489) uses assembly
       - INLINE ASM (MindX.sol#482-485)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #assembly-usage
INFO:Detectors:
Mindx._transfer(address,address,uint256) (MindX.sol#815-880) compares to
   ↪  a boolean constant:
       -tradingEnabled == false (MindX.sol#840)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #boolean-equality
INFO:Detectors:
Different versions of Solidity are used:
       - Version used: ['^0.8.0', '^0.8.20']
       - ^0.8.0 (MindX.sol#138)
       - ^0.8.20 (MindX.sol#7)
       - ^0.8.20 (MindX.sol#38)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #different-pragma-directives-are-used
INFO:Detectors:
Address._revert(bytes,string) (MindX.sol#474-489) is never used and
   ↪ should be removed
Address.functionCall(address,bytes) (MindX.sol#333-344) is never used
   ↪ and should be removed
Address.functionCall(address,bytes,string) (MindX.sol#346-352) is never
   ↪ used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MindX.sol#354-366)
   ↪  is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (MindX.sol
   ↪ #368-388) is never used and should be removed
Address.functionDelegateCall(address,bytes) (MindX.sol#417-427) is never
   ↪  used and should be removed
Address.functionDelegateCall(address,bytes,string) (MindX.sol#429-442)
   ↪ is never used and should be removed
Address.functionStaticCall(address,bytes) (MindX.sol#390-400) is never
   ↪ used and should be removed
Address.functionStaticCall(address,bytes,string) (MindX.sol#402-415) is
   ↪ never used and should be removed
Address.isContract(address) (MindX.sol#316-318) is never used and should
   ↪  be removed
Address.sendValue(address,uint256) (MindX.sol#320-331) is never used and
   ↪  should be removed
```

```
Address.verifyCallResult(bool,bytes,string) (MindX.sol#462-472) is never
    ↪  used and should be removed
Address.verifyCallResultFromTarget(address,bool,bytes,string) (MindX.sol
    ↪ #444-460) is never used and should be removed
Context._contextSuffixLength() (MindX.sol#28-30) is never used and
    ↪ should be removed
Context._msgData() (MindX.sol#24-26) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code
INFO:Detectors:
Pragma version^0.8.20 (MindX.sol#7) necessitates a version too recent to
    ↪  be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.20 (MindX.sol#38) necessitates a version too recent
    ↪ to be trusted. Consider deploying with 0.8.18.
Pragma version^0.8.0 (MindX.sol#138) allows old versions
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 Address.sendValue(address,uint256) (MindX.sol#320-331)
    ↪ :
        - (success) = recipient.call{value: amount}() (MindX.sol#329)
Low level call in Address.functionCallWithValue(address,bytes,uint256,
    ↪ string) (MindX.sol#368-388):
        - (success,returndata) = target.call{value: value}(data) (MindX.
            ↪ sol#378-380)
Low level call in Address.functionStaticCall(address,bytes,string) (
    ↪ MindX.sol#402-415):
        - (success,returndata) = target.staticcall(data) (MindX.sol#407)
Low level call in Address.functionDelegateCall(address,bytes,string) (
    ↪ MindX.sol#429-442):
        - (success,returndata) = target.delegatecall(data) (MindX.sol
            ↪ #434)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #low-level-calls
INFO:Detectors:
Function IUniswapV2Router01.WETH() (MindX.sol#143) is not in mixedCase
Event Mindx.adding_isExcluded(address) (MindX.sol#718) is not in
    ↪ CapWords
Event Mindx.removing_isExcluded(address) (MindX.sol#719) is not in
    ↪ CapWords
Event Mindx.adding_automated(address) (MindX.sol#720) is not in CapWords
Event Mindx.removing_automated(address) (MindX.sol#721) is not in
    ↪ CapWords
Event Mindx.enable_trading(bool) (MindX.sol#722) is not in CapWords
Event Mindx.tax_change(uint256,uint256) (MindX.sol#723) is not in
    ↪ CapWords
Event Mindx.tax_Treasury(address,address) (MindX.sol#724) is not in
    ↪ CapWords
Event Mindx.tax_fee(uint256,uint256) (MindX.sol#725) is not in CapWords
Parameter Mindx.enableTrading(bool)._status (MindX.sol#775) is not in
    ↪ mixedCase
Parameter Mindx.taxChange(uint256,uint256)._b (MindX.sol#781) is not in
    ↪ mixedCase
Parameter Mindx.taxChange(uint256,uint256)._s (MindX.sol#781) is not in
    ↪ mixedCase
Parameter Mindx.divAdress(address,address)._tr (MindX.sol#802) is not in
    ↪  mixedCase
Parameter Mindx.divAdress(address,address)._to (MindX.sol#802) is not in
    ↪  mixedCase
Function Mindx.adding_isExcludedMaxTransactionAmount(address) (MindX.sol
    ↪ #919-925) is not in mixedCase
Parameter Mindx.adding_isExcludedMaxTransactionAmount(address)._a (MindX
    ↪ .sol#919) is not in mixedCase
Function Mindx.removing_isExcludedMaxTransactionAmount(address) (MindX.
    ↪ sol#927-930) is not in mixedCase
```

```
Parameter Mindx.removing_isExcludedMaxTransactionAmount(address)._a (
    ↪ MindX.sol#927) is not in mixedCase
Function Mindx.adding_automatedMarketMakerPairs(address) (MindX.sol
    ↪ #932-935) is not in mixedCase
Parameter Mindx.adding_automatedMarketMakerPairs(address)._a (MindX.sol
    ↪ #932) is not in mixedCase
Function Mindx.removing_automatedMarketMakerPairs(address) (MindX.sol
    ↪ #937-940) is not in mixedCase
Parameter Mindx.removing_automatedMarketMakerPairs(address)._a (MindX.
    ↪ sol#937) is not in mixedCase
Function Mindx.BurnDevToken(uint256) (MindX.sol#946-948) is not in
    ↪ mixedCase
Parameter Mindx.setSwapAndLiquifyEnabled(bool)._enabled (MindX.sol#950)
    ↪ is not in mixedCase
Variable Mindx._isExcludedMaxTransactionAmount (MindX.sol#681) is not in
    ↪  mixedCase
Variable Mindx._automatedMarketMaker (MindX.sol#682) is not in mixedCase
Variable Mindx.RevenueShare (MindX.sol#685) is not in mixedCase
Variable Mindx.OwnerShare (MindX.sol#686) is not in mixedCase
Variable Mindx.initial_tax (MindX.sol#689) is not in mixedCase
Variable Mindx.second_buy_limit (MindX.sol#695) is not in mixedCase
Variable Mindx.initial_inject_timestamp (MindX.sol#696) is not in
    ↪ mixedCase
Variable Mindx.TechTeam (MindX.sol#697) is not in mixedCase
Variable Mindx.TreasuryRevenue (MindX.sol#698) is not in mixedCase
Variable Mindx.TreasuryOwner (MindX.sol#699) is not in mixedCase
Variable Mindx.Marketing (MindX.sol#700) is not in mixedCase
Variable Mindx.CEX (MindX.sol#701) is not in mixedCase
Variable Mindx.PreSale (MindX.sol#702) is not in mixedCase
Variable Mindx.CReward (MindX.sol#703) is not in mixedCase
Variable Mindx.Surplus (MindX.sol#704) is not in mixedCase
Variable Mindx._tierTimestamp (MindX.sol#706) is not in mixedCase
Variable Mindx._transactorLastblock (MindX.sol#708) is not in mixedCase
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #conformance-to-solidity-naming-conventions
INFO:Detectors:
Reentrancy in Mindx._transfer(address,address,uint256) (MindX.sol
   ↪ #815-880):
       External calls:
       - swapAndReward(contractTokenBalance) (MindX.sol#833)
              - address(TreasuryOwner).transfer(newBalance) (MindX.sol
                 ↪ #895)
       State variables written after the call(s):
       - super._transfer(from,TreasuryOwner,(amount / 100) * initial_tax
          ↪ ) (MindX.sol#865)
              - _balances[sender] = senderBalance - amount (MindX.sol
                 ↪ #616)
              - _balances[recipient] += amount (MindX.sol#618)
       - super._transfer(from,address(this),Taxation) (MindX.sol#873)
              - _balances[sender] = senderBalance - amount (MindX.sol
                 ↪ #616)
              - _balances[recipient] += amount (MindX.sol#618)
       - super._transfer(from,to,amount - Taxation) (MindX.sol#879)
              - _balances[sender] = senderBalance - amount (MindX.sol
                 ↪ #616)
              - _balances[recipient] += amount (MindX.sol#618)
       - _tierTimestamp[to] = block.timestamp (MindX.sol#876)
       - _tierTimestamp[from] = block.timestamp (MindX.sol#877)
       - _transactorLastblock[tx.origin] = block.number (MindX.sol#867)
       - isBot[from] = true (MindX.sol#858)
       - isBot[to] = true (MindX.sol#860)
       Event emitted after the call(s):
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(from,to,amount - Taxation) (MindX.sol
                 ↪ #879)
       - Transfer(sender,recipient,amount) (MindX.sol#620)
```

```
              - super._transfer(from,address(this),Taxation) (MindX.sol
                 ↪ #873)
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(from,TreasuryOwner,(amount / 100) *
                 ↪ initial_tax) (MindX.sol#865)
Reentrancy in Mindx.swapAndReward(uint256) (MindX.sol#882-901):
       External calls:
       - address(TreasuryOwner).transfer(newBalance) (MindX.sol#895)
       State variables written after the call(s):
       - super._transfer(address(this),TreasuryRevenue,_revenue_share) (
          ↪ MindX.sol#898)
              - _balances[sender] = senderBalance - amount (MindX.sol
                 ↪ #616)
              - _balances[recipient] += amount (MindX.sol#618)
       - super._transfer(address(this),TreasuryOwner,_owner_share) (
          ↪ MindX.sol#899)
              - _balances[sender] = senderBalance - amount (MindX.sol
                 ↪ #616)
              - _balances[recipient] += amount (MindX.sol#618)
       Event emitted after the call(s):
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(address(this),TreasuryOwner,_owner_share
                 ↪ ) (MindX.sol#899)
       - Transfer(sender,recipient,amount) (MindX.sol#620)
              - super._transfer(address(this),TreasuryRevenue,
                 ↪ _revenue_share) (MindX.sol#898)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-4
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256
   ↪ ,uint256,uint256,address,uint256).amountADesired (MindX.sol#148)
   ↪ is too similar to IUniswapV2Router01.addLiquidity(address,address
   ↪ ,uint256,uint256,uint256,uint256,address,uint256).amountBDesired
   ↪ (MindX.sol#149)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #variable-names-too-similar
INFO:Detectors:
Mindx.swapping (MindX.sol#687) is never used in Mindx (MindX.sol
    ↪ #678-954)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-state-variable
INFO:Detectors:
Mindx.OwnerShare (MindX.sol#686) should be constant
Mindx.RevenueShare (MindX.sol#685) should be constant
Mindx.initial_tax (MindX.sol#689) should be constant
Mindx.numTokensSellToShareRevenue (MindX.sol#694) should be constant
Mindx.second_buy_limit (MindX.sol#695) should be constant
Mindx.swapping (MindX.sol#687) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-constant
INFO:Detectors:
Mindx.uniswapV2Router (MindX.sol#693) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-immutable
INFO:Slither:. analyzed (9 contracts with 94 detectors), 96 result(s)
    ↪ found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

We examined the design and implementation of MindX in this audit and found several issues of various severities. We advise mindx team to implement the recommendations contained in all 13 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

BLOCKHAT

SECURITY

For a Smart Contract Audit, contact us at contact@blockhat.io