auditace.tech

AuditAce

FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

AiSora

DATED : 26 Feb, 2024

MANUAL TESTING

Centralization – Enabling Trades

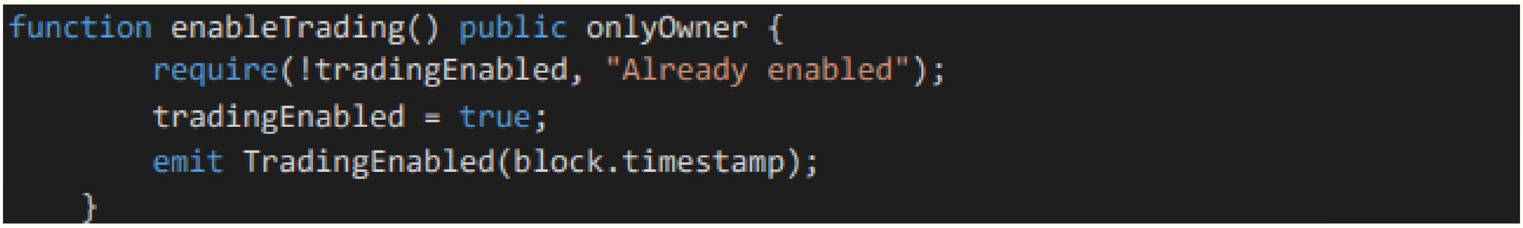
Severity: High

Function: EnableTrading

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 enableTrading() public onlyOwner {
        require(!tradingEnabled, "Already enabled");
        tradingEnabled = true;
        emit TradingEnabled(block.timestamp);
    }
```

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.

AUDIT SUMMARY

Project name – AiSora

Date: 26 Feb, 2024

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: **Passed With High Risk**

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	1	0	1	3
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :
The code has undergone static analysis using Slither.

Testnet version:
The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:
https://testnet.bscscan.com/address/0xafa30a219e4cee216e6cb350b7afb25f69689c4a#code

Token Information

Token Name : AiSora

Token Symbol: AiSora

Decimals: 18

Token Supply: 900,000,000

Network: BscScan

Token Type: BEP-20

Token Address:
0x87dfbC5926239760CEEb6f3C57199ba2358a524b

Checksum:
B67acbefe2a12642d388659dffd20712

Owner:
0xaD0705f93aeEE52adE5a95E54B52bB47c62Cc5F0
(at time of writing the audit)

Deployer:
0x5240fA85dE586c651245ABfE17D716B72f02f6F3

TOKEN OVERVIEW

Fees:

Buy Fee: 5%

Sell Fee: 5%

Transfer Fee: 0-0%

Fees Privilege: Owner

Ownership: Owned

Minting: No mint function

Max Tx Amount/ Max Wallet Amount: No

Blacklist: No

Other Privileges:

-Whitelist to transfer without enabling trades

- Enabling trades

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.

- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.

- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.

- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.

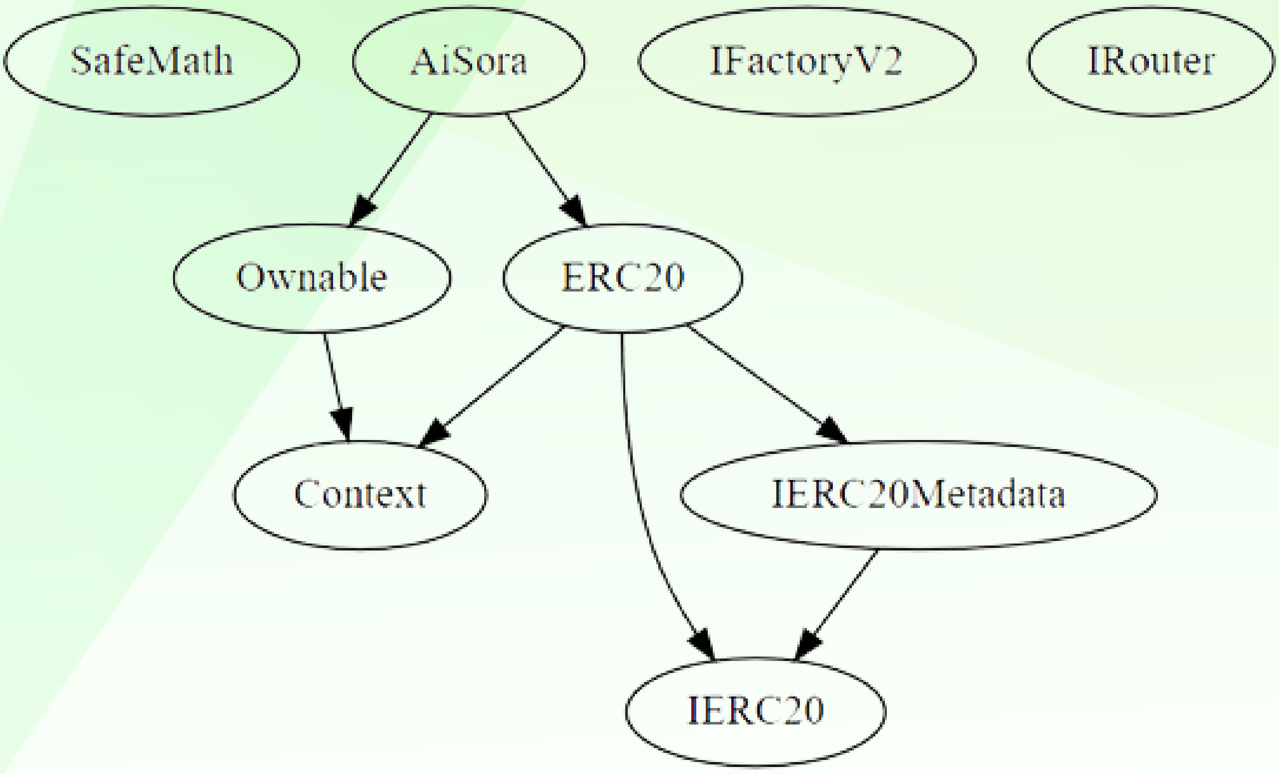
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.

- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- ✓ Return values of low-level calls
- ✓ Private modifier
- ✓ Multiple Sends
- ✓ Using Suicide
- ✓ Gas Limitand Loops
- ✓ Address hardcoded
- ✓ Exception Disorder
- ✓ Using inline assembly
- ✓ Divide before multiply
- ✓ Missing Zero Address Validation
- ✓ Compiler version not fixed
- ✓ **Gasless Send**
- ✓ Using block.timestamp
- ✓ Re-entrancy
- ✓ Tautology or contradiction
- ✓ Timestamp Dependence
- ✓ Revert/require functions
- ✓ Use of tx.origin
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Using SHA3
- ✓ Using throw

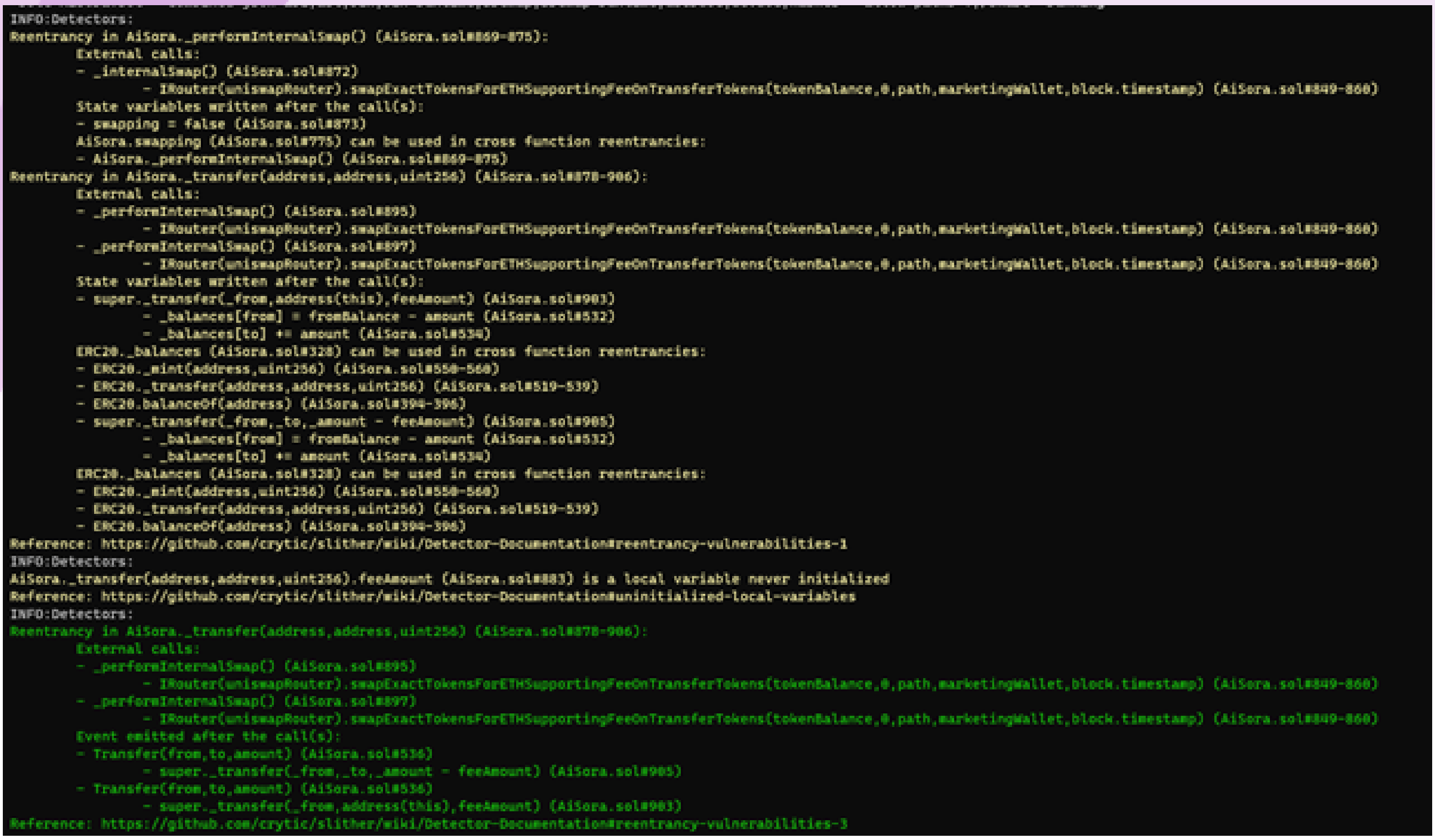
INHERITANCE TREE



STATIC ANALYSIS

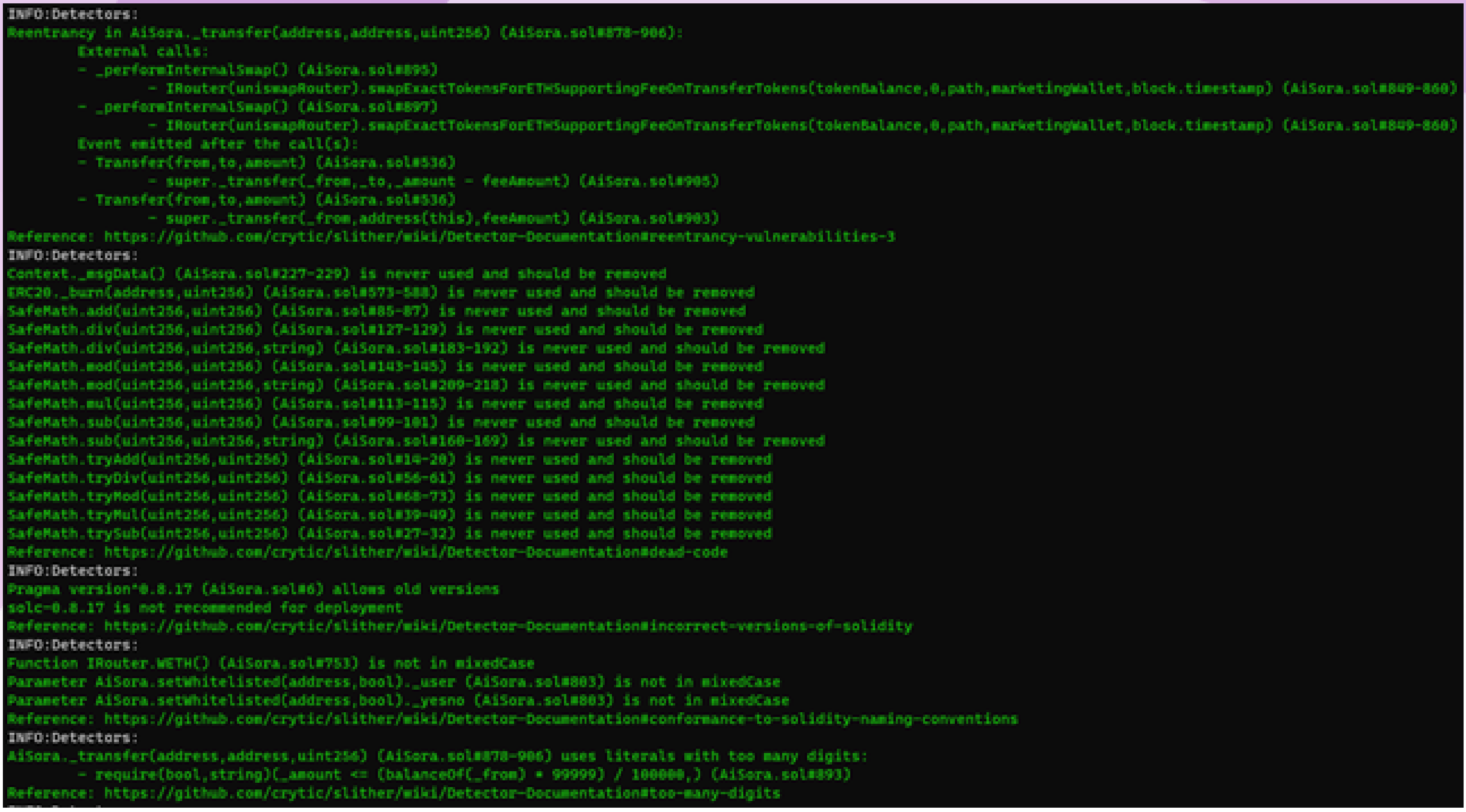
A static analysis of the code was performed using Slither.

No issues were found.

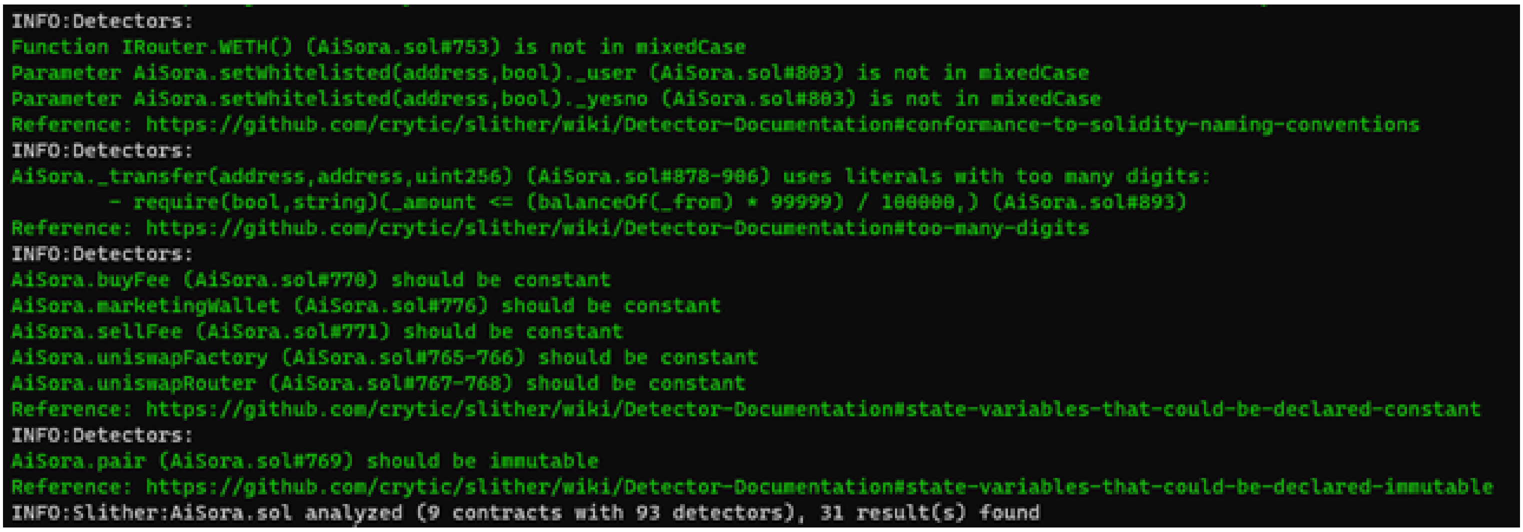


```
INFO:Detectors:
Reentrancy in AiSora._performInternalSwap() (AiSora.sol#869-875):
        External calls:
        - _internalSwap() (AiSora.sol#872)
                - IRouter(uniswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (AiSora.sol#849-860)
        State variables written after the call(s):
        - swapping = false (AiSora.sol#873)
        AiSora.swapping (AiSora.sol#775) can be used in cross function reentrancies:
        - AiSora._performInternalSwap() (AiSora.sol#869-875)
Reentrancy in AiSora._transfer(address,address,uint256) (AiSora.sol#878-906):
        External calls:
        - _performInternalSwap() (AiSora.sol#895)
                - IRouter(uniswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (AiSora.sol#849-860)
        - _performInternalSwap() (AiSora.sol#897)
                - IRouter(uniswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (AiSora.sol#849-860)
        State variables written after the call(s):
        - super._transfer(_from,address(this),feeAmount) (AiSora.sol#903)
                - _balances[from] = fromBalance - amount (AiSora.sol#532)
                - _balances[to] += amount (AiSora.sol#534)
        ERC20._balances (AiSora.sol#328) can be used in cross function reentrancies:
        - ERC20._mint(address,uint256) (AiSora.sol#550-560)
        - ERC20._transfer(address,address,uint256) (AiSora.sol#519-539)
        - ERC20.balanceOf(address) (AiSora.sol#394-396)
        - super._transfer(_from,_to,_amount - feeAmount) (AiSora.sol#905)
                - _balances[from] = fromBalance - amount (AiSora.sol#532)
                - _balances[to] += amount (AiSora.sol#534)
        ERC20._balances (AiSora.sol#328) can be used in cross function reentrancies:
        - ERC20._mint(address,uint256) (AiSora.sol#550-560)
        - ERC20._transfer(address,address,uint256) (AiSora.sol#519-539)
        - ERC20.balanceOf(address) (AiSora.sol#394-396)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
AiSora._transfer(address,address,uint256).feeAmount (AiSora.sol#883) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
Reentrancy in AiSora._transfer(address,address,uint256) (AiSora.sol#878-906):
        External calls:
        - _performInternalSwap() (AiSora.sol#895)
                - IRouter(uniswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (AiSora.sol#849-860)
        - _performInternalSwap() (AiSora.sol#897)
                - IRouter(uniswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (AiSora.sol#849-860)
        Event emitted after the call(s):
        - Transfer(from,to,amount) (AiSora.sol#536)
                - super._transfer(_from,_to,_amount - feeAmount) (AiSora.sol#905)
        - Transfer(from,to,amount) (AiSora.sol#536)
                - super._transfer(_from,address(this),feeAmount) (AiSora.sol#903)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

STATIC ANALYSIS



```
INFO:Detectors:
Reentrancy in AiSora._transfer(address,address,uint256) (AiSora.sol#878-906):
        External calls:
        - _performInternalSwap() (AiSora.sol#895)
                - IRouter(uniswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (AiSora.sol#849-860)
        - _performInternalSwap() (AiSora.sol#897)
                - IRouter(uniswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (AiSora.sol#849-860)
        Event emitted after the call(s):
        - Transfer(from,to,amount) (AiSora.sol#536)
                - super._transfer(_from,_to,_amount - feeAmount) (AiSora.sol#905)
        - Transfer(from,to,amount) (AiSora.sol#536)
                - super._transfer(_from,address(this),feeAmount) (AiSora.sol#903)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Context._msgData() (AiSora.sol#227-229) is never used and should be removed
ERC20._burn(address,uint256) (AiSora.sol#573-588) is never used and should be removed
SafeMath.add(uint256,uint256) (AiSora.sol#85-87) is never used and should be removed
SafeMath.div(uint256,uint256) (AiSora.sol#127-129) is never used and should be removed
SafeMath.div(uint256,uint256,string) (AiSora.sol#183-192) is never used and should be removed
SafeMath.mod(uint256,uint256) (AiSora.sol#143-145) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (AiSora.sol#209-218) is never used and should be removed
SafeMath.mul(uint256,uint256) (AiSora.sol#113-115) is never used and should be removed
SafeMath.sub(uint256,uint256) (AiSora.sol#99-101) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (AiSora.sol#160-169) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (AiSora.sol#14-20) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (AiSora.sol#56-61) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (AiSora.sol#68-73) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (AiSora.sol#39-49) is never used and should be removed
SafeMath.trySub(uint256,uint256) (AiSora.sol#27-32) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.17 (AiSora.sol#6) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function IRouter.WETH() (AiSora.sol#753) is not in mixedCase
Parameter AiSora.setWhitelisted(address,bool)._user (AiSora.sol#803) is not in mixedCase
Parameter AiSora.setWhitelisted(address,bool)._yesno (AiSora.sol#803) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
AiSora._transfer(address,address,uint256) (AiSora.sol#878-906) uses literals with too many digits:
        - require(bool,string)(_amount <= (balanceOf(_from) * 99999) / 100000,) (AiSora.sol#893)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```



```
INFO:Detectors:
Function IRouter.WETH() (AiSora.sol#753) is not in mixedCase
Parameter AiSora.setWhitelisted(address,bool)._user (AiSora.sol#803) is not in mixedCase
Parameter AiSora.setWhitelisted(address,bool)._yesno (AiSora.sol#803) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
AiSora._transfer(address,address,uint256) (AiSora.sol#878-906) uses literals with too many digits:
        - require(bool,string)(_amount <= (balanceOf(_from) * 99999) / 100000,) (AiSora.sol#893)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
AiSora.buyFee (AiSora.sol#770) should be constant
AiSora.marketingWallet (AiSora.sol#776) should be constant
AiSora.sellFee (AiSora.sol#771) should be constant
AiSora.uniswapFactory (AiSora.sol#765-766) should be constant
AiSora.uniswapRouter (AiSora.sol#767-768) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
AiSora.pair (AiSora.sol#769) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:AiSora.sol analyzed (9 contracts with 93 detectors), 31 result(s) found
```

FUNCTIONAL TESTING

1- Approve (passed):
https://testnet.bscscan.com/tx/0xe912079adab63d539bb390f1b6039d919ab230e1de6c3fb8d5881832057ed007

2- Increase Allowance (passed):
https://testnet.bscscan.com/tx/0xf3e5e9368651968e96f27bc29d0ff653bca960b862f85f61dd40eb20530db9a0

3- Decrease Allowance (passed):
https://testnet.bscscan.com/tx/0x8e78960e3567ec2fed42d34e1f8f4a7ef226ff930c7c7c7a333ec1b6517fce0f

4- Enable Trading (passed):
https://testnet.bscscan.com/tx/0x170ee0c0648b8394c9e73c137e69e559fe50a245ebeea6d2bf3269c3ce7d1602

POINTS TO NOTE

- The owner can transfer ownership.
- The owner can renounce ownership.
- The owner can Enable trading.
- The owner can set a whitelisted address.
- The owner can remove the maximum wallet limit.

CLASSIFICATION OF RISK

Severity	Description
◆ Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
◆ High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
◆ Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
◆ Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
◆ Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
◆ Critical	0
◆ High-Risk	1
◆ Medium-Risk	0
◆ Low-Risk	1
◆ Gas Optimization / Suggestions	3

MANUAL TESTING

Centralization – Enabling Trades

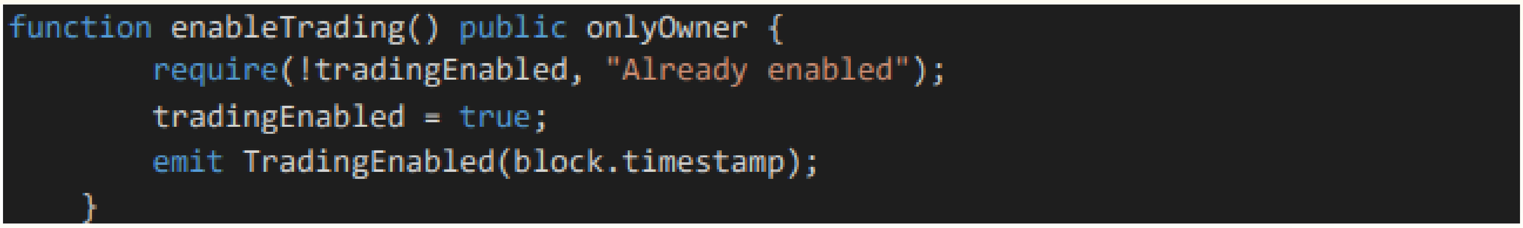
Severity: High

Function: EnableTrading

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 enableTrading() public onlyOwner {
        require(!tradingEnabled, "Already enabled");
        tradingEnabled = true;
        emit TradingEnabled(block.timestamp);
    }
```

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.

MANUAL TESTING

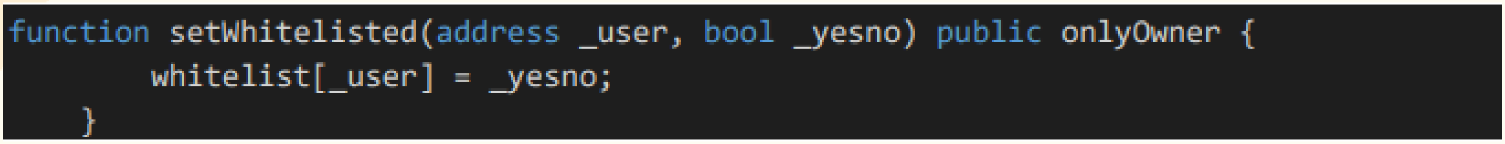
Centralization – Missing Events

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 setWhitelisted(address _user, bool _yesno) public onlyOwner {
        whitelist[_user] = _yesno;
    }
```

MANUAL TESTING

Optimization

Severity: Informational

Subject: Remove Safe Math

Status: Open

Line: 8-129

Overview:
compiler version above 0.8.0 can control arithmetic overflow/underflow, it is recommended to remove the unwanted code to avoid high gas fees.

MANUAL TESTING

Optimization

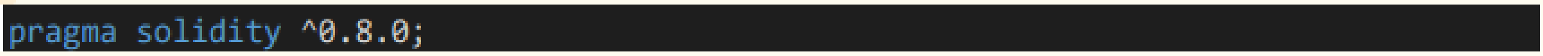
Severity: Informational

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.

MANUAL TESTING

Optimization

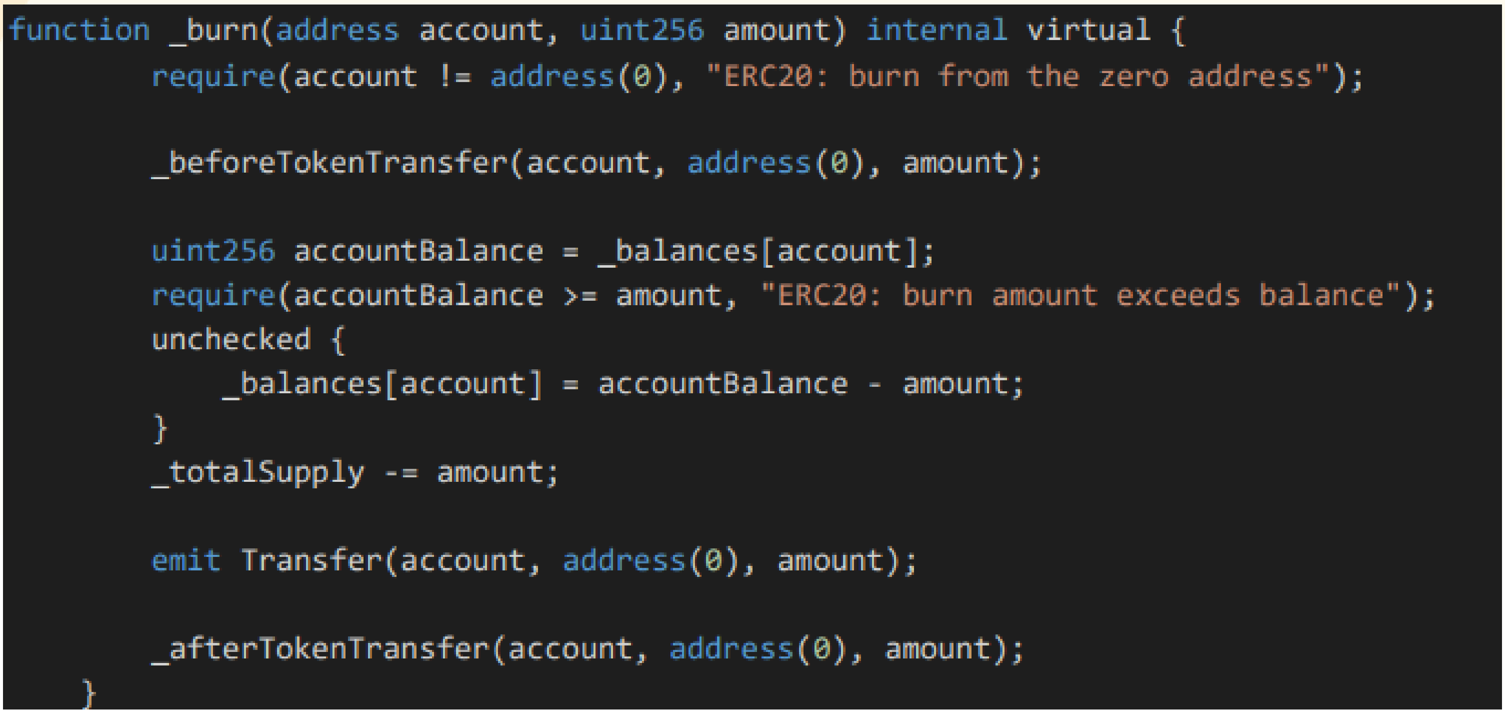
Severity: Optimization

Subject: Remove unused code

Status: Open

Overview:

Unused variables are allowed in Solidity, and they do. not pose a direct security issue. It is the best practice. though to avoid them.



```
function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
        }
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }
```

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

🌐 https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace