auditace.tech

AuditAce

FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

ACEEAGLE

DATED : 28 Nov 23'

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() external onlyOwner{
   require(tradingEnabled == false, "Trading is already enabled");
   tradingEnabled = 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 provide investors with more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad faith actions by the original owner.

MANUAL TESTING

Centralization – Buy and Sell fees.

Severity: High

function: setBuyFeePercentages

Status: Open

Overview:

The owner can set the buy and sell fees to more than 100%, which is not recommended.

```
function setBuyFeePercentages(uint256 _taxFeeonBuy, uint256
_liquidityFeeonBuy, uint256 _marketingFeeonBuy, uint256
_burnFeeOnBuy) external onlyOwner {
    taxFeeonBuy = _taxFeeonBuy;
    liquidityFeeonBuy = _liquidityFeeonBuy;
    marketingFeeonBuy = _marketingFeeonBuy;
    burnFeeOnBuy = _burnFeeOnBuy;

    totalBuyFees = taxFeeonBuy + liquidityFeeonBuy +
marketingFeeonBuy + burnFeeOnBuy;

    require(totalBuyFees <= 100, "Buy fees cannot be greater than
10%");

    emit BuyFeesChanged(taxFeeonBuy, liquidityFeeonBuy,
marketingFeeonBuy);
  }
```

Suggestion

It is recommended that no fees in the contract should be more than 25% of the contract.

AUDIT SUMMARY

Project name – ACEEAGLE

Date: 28 Nov, 2023

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	2	1	2	1
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/0x18186c2828f280d2c5fba5786b85a59c104b33ab#code

Token Information

Token Address: 0x560d9a8beaae8b1bffeea1fc6ecb1f32dfb9495e

Name: ACEEAGLE

Symbol: AEA

Decimals: 9

Network: Etherscan

Token Type: ERC20

Owner: 0x12528AEa79914bd10a4b9f320358c905462339c1

Deployer: 0x12528AEa79914bd10a4b9f320358c905462339c1

Token Supply: 1000000000000000000

Checksum: cfe3cef7c2c788bc03532d7342fc9fae

Testnet: https://testnet.bscscan.com/address/0x18186c2828f280d2c5fba5786b85a59c104b33ab#code

TOKEN OVERVIEW

Buy Fee: 0-100%

Sell Fee: 0-100%

Transfer Fee: 0-0%

Fee Privilege: Owner

Ownership: Owned

Minting: None

Max Tx: Yes

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

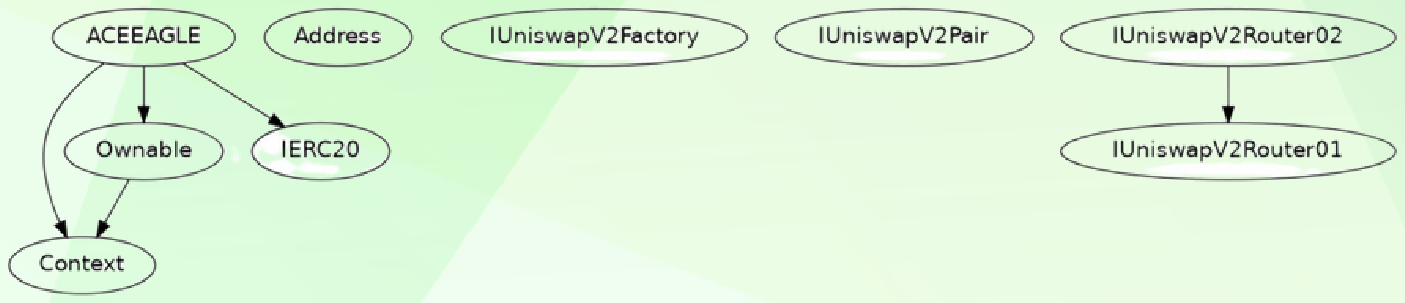
CLASSIFICATION OF RISK

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**	2
◆ **Medium-Risk**	1
◆ **Low-Risk**	2
◆ **Gas Optimization / Suggestions**	1

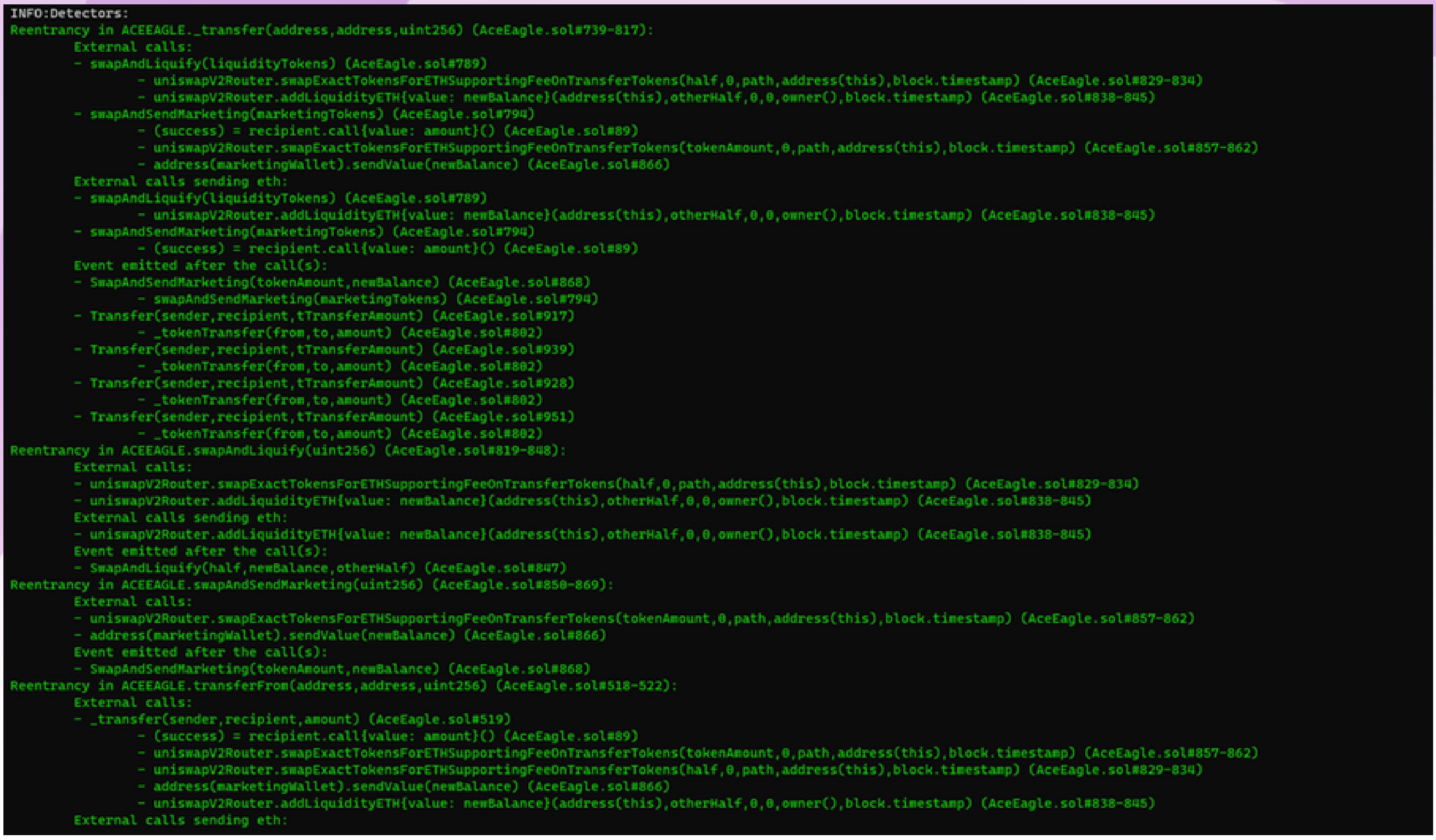
INHERITANCE TREE



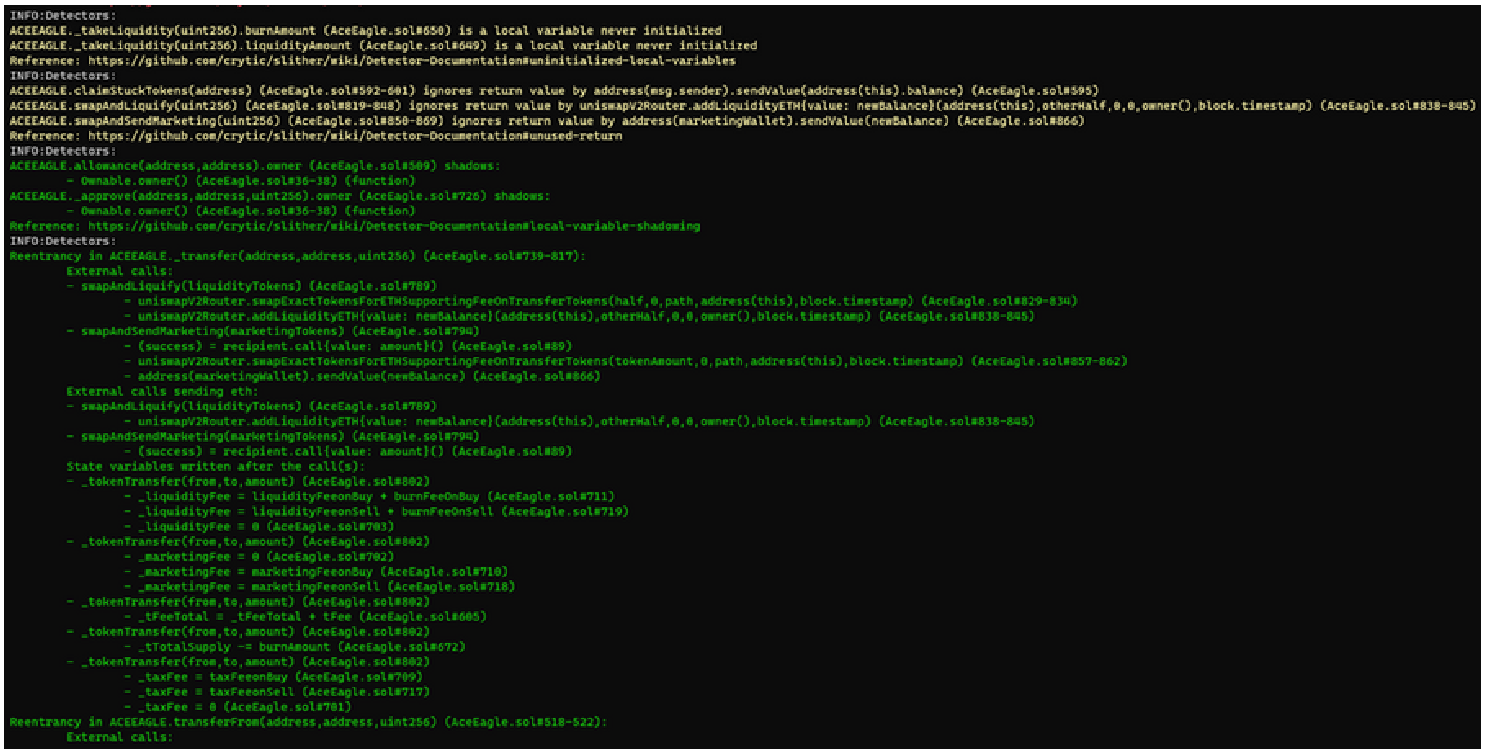
POINTS TO NOTE

- Owner can renounce ownership.
- Owner can transfer the ownership.
- Owner can exclude/include accounts from rewards.
- Owner can set swap tokens.
- Owner can set swap enable.
- Owner can set fees more than 100%
- Owner can enable max transactiion limit.
- Owner can set mmax wallet amount.

STATIC ANALYSIS

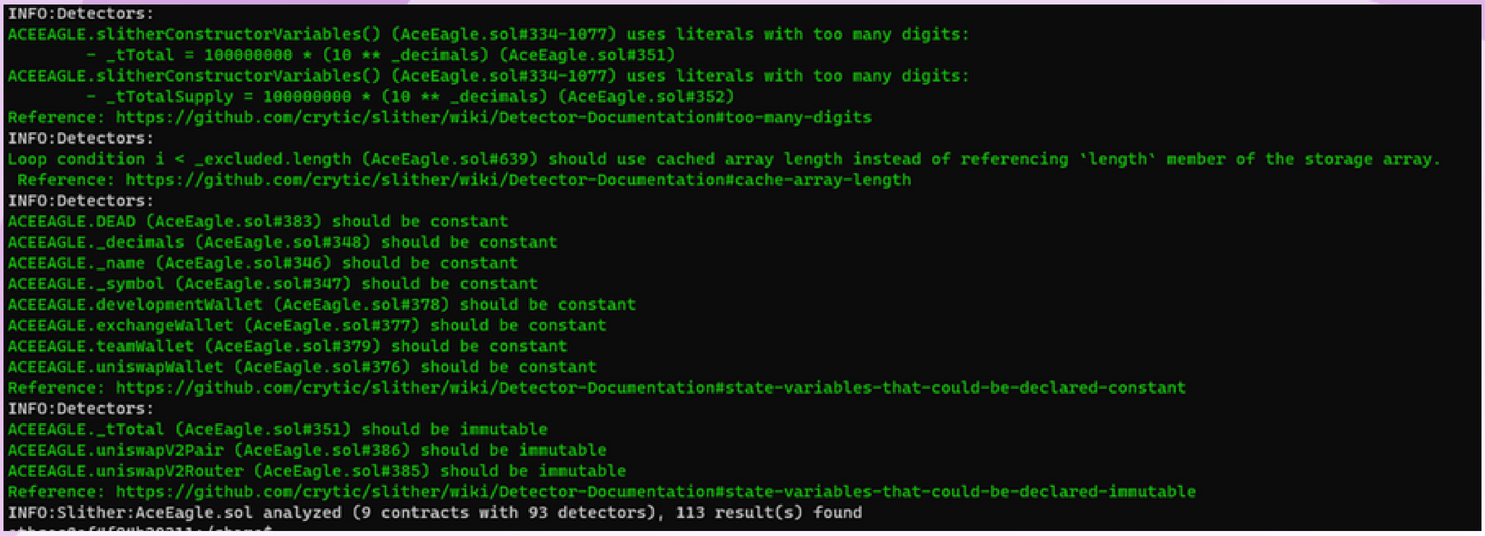


```
INFO:Detectors:
Reentrancy in ACEEAGLE._transfer(address,address,uint256) (AceEagle.sol#739-817):
        External calls:
        - swapAndLiquify(liquidityTokens) (AceEagle.sol#789)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(half,0,path,address(this),block.timestamp) (AceEagle.sol#829-834)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,owner(),block.timestamp) (AceEagle.sol#838-845)
        - swapAndSendMarketing(marketingTokens) (AceEagle.sol#794)
                - (success) = recipient.call{value: amount}() (AceEagle.sol#89)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (AceEagle.sol#857-862)
                - address(marketingWallet).sendValue(newBalance) (AceEagle.sol#866)
        External calls sending eth:
        - swapAndLiquify(liquidityTokens) (AceEagle.sol#789)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,owner(),block.timestamp) (AceEagle.sol#838-845)
        - swapAndSendMarketing(marketingTokens) (AceEagle.sol#794)
                - (success) = recipient.call{value: amount}() (AceEagle.sol#89)
        Event emitted after the call(s):
        - SwapAndSendMarketing(tokenAmount,newBalance) (AceEagle.sol#868)
                - swapAndSendMarketing(marketingTokens) (AceEagle.sol#794)
        - Transfer(sender,recipient,tTransferAmount) (AceEagle.sol#917)
                - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
        - Transfer(sender,recipient,tTransferAmount) (AceEagle.sol#939)
                - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
        - Transfer(sender,recipient,tTransferAmount) (AceEagle.sol#928)
                - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
        - Transfer(sender,recipient,tTransferAmount) (AceEagle.sol#951)
                - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
Reentrancy in ACEEAGLE.swapAndLiquify(uint256) (AceEagle.sol#819-848):
        External calls:
        - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(half,0,path,address(this),block.timestamp) (AceEagle.sol#829-834)
        - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,owner(),block.timestamp) (AceEagle.sol#838-845)
        External calls sending eth:
        - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,owner(),block.timestamp) (AceEagle.sol#838-845)
        Event emitted after the call(s):
        - SwapAndLiquify(half,newBalance,otherHalf) (AceEagle.sol#847)
Reentrancy in ACEEAGLE.swapAndSendMarketing(uint256) (AceEagle.sol#850-869):
        External calls:
        - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (AceEagle.sol#857-862)
        - address(marketingWallet).sendValue(newBalance) (AceEagle.sol#866)
        Event emitted after the call(s):
        - SwapAndSendMarketing(tokenAmount,newBalance) (AceEagle.sol#868)
Reentrancy in ACEEAGLE.transferFrom(address,address,uint256) (AceEagle.sol#518-522):
        External calls:
        - _transfer(sender,recipient,amount) (AceEagle.sol#519)
                - (success) = recipient.call{value: amount}() (AceEagle.sol#89)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (AceEagle.sol#857-862)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(half,0,path,address(this),block.timestamp) (AceEagle.sol#829-834)
                - address(marketingWallet).sendValue(newBalance) (AceEagle.sol#866)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,owner(),block.timestamp) (AceEagle.sol#838-845)
        External calls sending eth:
```



```
INFO:Detectors:
ACEEAGLE._takeLiquidity(uint256).burnAmount (AceEagle.sol#650) is a local variable never initialized
ACEEAGLE._takeLiquidity(uint256).liquidityAmount (AceEagle.sol#649) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ACEEAGLE.claimStuckTokens(address) (AceEagle.sol#592-601) ignores return value by address(msg.sender).sendValue(address(this).balance) (AceEagle.sol#595)
ACEEAGLE.swapAndLiquify(uint256) (AceEagle.sol#819-848) ignores return value by uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,owner(),block.timestamp) (AceEagle.sol#838-845)
ACEEAGLE.swapAndSendMarketing(uint256) (AceEagle.sol#850-869) ignores return value by address(marketingWallet).sendValue(newBalance) (AceEagle.sol#866)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
ACEEAGLE.allowance(address,address).owner (AceEagle.sol#509) shadows:
        - Ownable.owner() (AceEagle.sol#36-38) (function)
ACEEAGLE._approve(address,address,uint256).owner (AceEagle.sol#726) shadows:
        - Ownable.owner() (AceEagle.sol#36-38) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in ACEEAGLE._transfer(address,address,uint256) (AceEagle.sol#739-817):
        External calls:
        - swapAndLiquify(liquidityTokens) (AceEagle.sol#789)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(half,0,path,address(this),block.timestamp) (AceEagle.sol#829-834)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,owner(),block.timestamp) (AceEagle.sol#838-845)
        - swapAndSendMarketing(marketingTokens) (AceEagle.sol#794)
                - (success) = recipient.call{value: amount}() (AceEagle.sol#89)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (AceEagle.sol#857-862)
                - address(marketingWallet).sendValue(newBalance) (AceEagle.sol#866)
        External calls sending eth:
        - swapAndLiquify(liquidityTokens) (AceEagle.sol#789)
                - uniswapV2Router.addLiquidityETH{value: newBalance}(address(this),otherHalf,0,0,owner(),block.timestamp) (AceEagle.sol#838-845)
        - swapAndSendMarketing(marketingTokens) (AceEagle.sol#794)
                - (success) = recipient.call{value: amount}() (AceEagle.sol#89)
        State variables written after the call(s):
        - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
                - _liquidityFee = liquidityFeeonBuy + burnFeeOnBuy (AceEagle.sol#711)
                - _liquidityFee = liquidityFeeonSell + burnFeeOnSell (AceEagle.sol#719)
                - _liquidityFee = 0 (AceEagle.sol#703)
        - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
                - _marketingFee = 0 (AceEagle.sol#702)
                - _marketingFee = marketingFeeonBuy (AceEagle.sol#710)
                - _marketingFee = marketingFeeonSell (AceEagle.sol#718)
        - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
                - _tFeeTotal = _tFeeTotal + tFee (AceEagle.sol#605)
        - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
                - _tTotalSupply -= burnAmount (AceEagle.sol#672)
        - _tokenTransfer(from,to,amount) (AceEagle.sol#802)
                - _taxFee = taxFeeonBuy (AceEagle.sol#709)
                - _taxFee = taxFeeonSell (AceEagle.sol#717)
                - _taxFee = 0 (AceEagle.sol#701)
Reentrancy in ACEEAGLE.transferFrom(address,address,uint256) (AceEagle.sol#518-522):
        External calls:
```

STATIC ANALYSIS



```
INFO:Detectors:
ACEEAGLE.slitherConstructorVariables() (AceEagle.sol#334-1077) uses literals with too many digits:
        - _tTotal = 100000000 * (10 ** _decimals) (AceEagle.sol#351)
ACEEAGLE.slitherConstructorVariables() (AceEagle.sol#334-1077) uses literals with too many digits:
        - _tTotalSupply = 100000000 * (10 ** _decimals) (AceEagle.sol#352)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
Loop condition i < _excluded.length (AceEagle.sol#639) should use cached array length instead of referencing `length` member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
INFO:Detectors:
ACEEAGLE.DEAD (AceEagle.sol#383) should be constant
ACEEAGLE._decimals (AceEagle.sol#348) should be constant
ACEEAGLE._name (AceEagle.sol#346) should be constant
ACEEAGLE._symbol (AceEagle.sol#347) should be constant
ACEEAGLE.developmentWallet (AceEagle.sol#378) should be constant
ACEEAGLE.exchangeWallet (AceEagle.sol#377) should be constant
ACEEAGLE.teamWallet (AceEagle.sol#379) should be constant
ACEEAGLE.uniswapWallet (AceEagle.sol#376) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
ACEEAGLE._tTotal (AceEagle.sol#351) should be immutable
ACEEAGLE.uniswapV2Pair (AceEagle.sol#386) should be immutable
ACEEAGLE.uniswapV2Router (AceEagle.sol#385) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:AceEagle.sol analyzed (9 contracts with 93 detectors), 113 result(s) found
```

Result => A static analysis of contract’s source code has been performed using slither,

No major issues were found in the output

FUNCTIONAL TESTING

1- Approve (passed):
https://testnet.bscscan.com/tx/0x0304f3eceb77878978e874d6117f4f1174fd7f48e7eb4c58dc91804c09463116

2- Increase Allowance (passed):
https://testnet.bscscan.com/tx/0x0f8ac1815f344efe6585349788146083bdf43c7d8151c10adeae7fe59ca78450

3- Decrease Allowance (passed):
https://testnet.bscscan.com/tx/0x2eeeab01f7dec31a6f0c2d0a04d493ec75108cd54868854c55d14d324b2de563

4- Enable Trading (passed):
https://testnet.bscscan.com/tx/0xc7f9be39e09022bce73f1ce2f70f9d78f0aa59eb3498cb8edc433e52b3b69515

5- Enable Wallet to Wallet Transfer Without Fee (passed):
https://testnet.bscscan.com/tx/0x12bd948d15891870307736856637707 51fd55b60c6cadf3ecaff5f833767e7d7

6- Set Swap Enabled (passed):
https://testnet.bscscan.com/tx/0xec773156ab8f3dddd6e6da98ddecdf9ed3eca4231b767541e4106af2d27b5b21

7- Change Marketing Wallet (passed):
https://testnet.bscscan.com/tx/0xaa48d853a746a2f3dba894044eba4ef25bf29fbabd3d3b8d9ef2bf599bd994a6

8- Transfer Ownership (passed):
https://testnet.bscscan.com/tx/0x76f36cd9ef64cc0775329f0f7c62acd00cdefe4c7d38469cdc1ce3e0d2ae5be4

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() external onlyOwner{
   require(tradingEnabled == false, "Trading is already enabled");
   tradingEnabled = 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 provide investors with more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad faith actions by the original owner.

MANUAL TESTING

Centralization – Buy and Sell fees.

Severity: High

function: setBuyFeePercentages

Status: Open

Overview:

The owner can set the buy and sell fees to more than 100%, which is not recommended.

```
function setBuyFeePercentages(uint256 _taxFeeonBuy, uint256
_liquidityFeeonBuy, uint256 _marketingFeeonBuy, uint256
_burnFeeOnBuy) external onlyOwner {
    taxFeeonBuy = _taxFeeonBuy;
    liquidityFeeonBuy = _liquidityFeeonBuy;
    marketingFeeonBuy = _marketingFeeonBuy;
    burnFeeOnBuy = _burnFeeOnBuy;

    totalBuyFees = taxFeeonBuy + liquidityFeeonBuy +
marketingFeeonBuy + burnFeeOnBuy;

    require(totalBuyFees <= 100, "Buy fees cannot be greater than
10%");

    emit BuyFeesChanged(taxFeeonBuy, liquidityFeeonBuy,
marketingFeeonBuy);
  }
```

Suggestion

It is recommended that no fees in the contract should be more than 25% of the contract.

MANUAL TESTING

Centralization – Liquidity is added to EOA.
Severity: Medium
function: swapAndLiquify
Status: Open

Overview:
Liquidity is adding to EOA. It may be drained by the addLiquidityETH.

```
 function swapAndLiquify(uint256 tokens) private {
    uint256 half = tokens / 2;
    uint256 otherHalf = tokens - half;

    uint256 initialBalance = address(this).balance;

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
      half,
      0, // accept any amount of ETH
      path,
      address(this),
      block.timestamp);

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

    uniswapV2Router.addLiquidityETH{value: newBalance}(
      address(this),
      otherHalf,
      0, // slippage is unavoidable
      0, // slippage is unavoidable
      owner(),
      block.timestamp
    );

    emit SwapAndLiquify(half, newBalance, otherHalf);
  }
```

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

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 openTrading() external onlyOwner {
    tradingOpen = true;
  }

 function setPreLaunchAddress(
    address _address,
    bool state
  ) external onlyOwner {
    preLaunchAddress[_address] = state;
  }
```

MANUAL TESTING

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), "ERC20: approve from the zero
address");
    require(spender != address(0), "ERC20: 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

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 functionCall(address target, bytes memory data) internal
returns (bytes memory) {
   return functionCall(target, data, "Address: low-level call failed");
  }

 function functionCall(address target, bytes memory data, string
memory errorMessage) internal returns (bytes memory) {
    return _functionCallWithValue(target, data, 0, errorMessage);
  }

 function functionCallWithValue(address target, bytes memory data,
uint256 value) internal returns (bytes memory) {
    return functionCallWithValue(target, data, value, "Address: low-
level call with value failed");
  }

 function functionCallWithValue(address target, bytes memory data,
uint256 value, string memory errorMessage) internal returns (bytes
memory) { //
    require(address(this).balance >= value, "Address: insufficient
balance for call");
```

MANUAL TESTING

```
 return _functionCallWithValue(target, data, value, errorMessage);
  }

  function _functionCallWithValue(address target, bytes memory
data, uint256 weiValue, string memory errorMessage) private
returns (bytes memory) {
    require(isContract(target), "Address: call to non-contract");

    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) = target.call{ value:
weiValue }(data);
    if (success) {
      return returndata;
    } else {
      // Look for revert reason and bubble it up if present
      if (returndata.length > 0) {
// The easiest way to bubble the revert reason is using memory via
assembly

// solhint-disable-next-line no-inline-assembly
assembly {
let returndata_size := mload(returndata)
revert(add(32, returndata), returndata_size)
        }
      } else {
revert(errorMessage);
      }
    }
  }
}
```

DISCLAIMER

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