Cyberscope

Audit Report

OpenSearchAI

May 2024

Network	ETH
Address	0x07EcdBf993Ed0925dd024C8FF1BD15871fcfee38
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved

Table of Contents

Review

Contract Name	OpenSearchAI
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://etherscan.io/address/0x07EcdBf993Ed0925dd024C8FF1BD15871fcfee38
Address	0x07EcdBf993Ed0925dd024C8FF1BD15871fcfee38
Network	ETH
Symbol	osAI
Decimals	18
Total Supply	1,000,000
Badge Eligibility	Yes

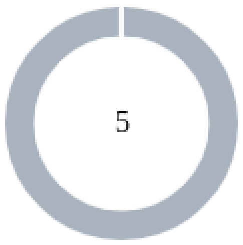
Audit Updates

Initial Audit	23 May 2024

Source Files

Filename	SHA256
OpenSearchAI.sol	7c169ef8298bbeb68650d76733b4f75fd36b05860fa9245c06eedd3a826f9ddc

Findings Breakdown



🔴	Critical	0
🟠	Medium	0
●	Minor / Informative	5

Severity		Unresolved	Acknowledged	Resolved	Other
🔴	Critical	0	0	0	0
🟠	Medium	0	0	0	0
●	Minor / Informative	5	0	0	0

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	OpenSearchAI.sol#L349
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
path[0] = address(this);         path[1] =
uniswapV2Router.WETH();

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
    contractTokenBalance,
    0,
    path,
    address(this),
    block.timestamp);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	OpenSearchAI.sol#L269,270,277,295
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
marketingWallet
stakingRevShareWallet
uniswapV2Pair
swapTokensAtAmount
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	OpenSearchAI.sol#L249,250,253,258
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public buyFee = 5
uint256 public sellFee = 5
uint256 private maxWalletLimit = 20
address private DEAD =
0x000000000000000000000000000000000000dEaD
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	OpenSearchAI.sol#L190
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
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];
...
        }
        _totalSupply -= amount;

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

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

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	OpenSearchAI.sol#L388
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
_isExcludedFromMaxWalletLimit[from]  == false &&
            _isExcludedFromMaxWalletLimit[to]    == false &&
            to != uniswapV2Pair && from == uniswapV2Pair
```

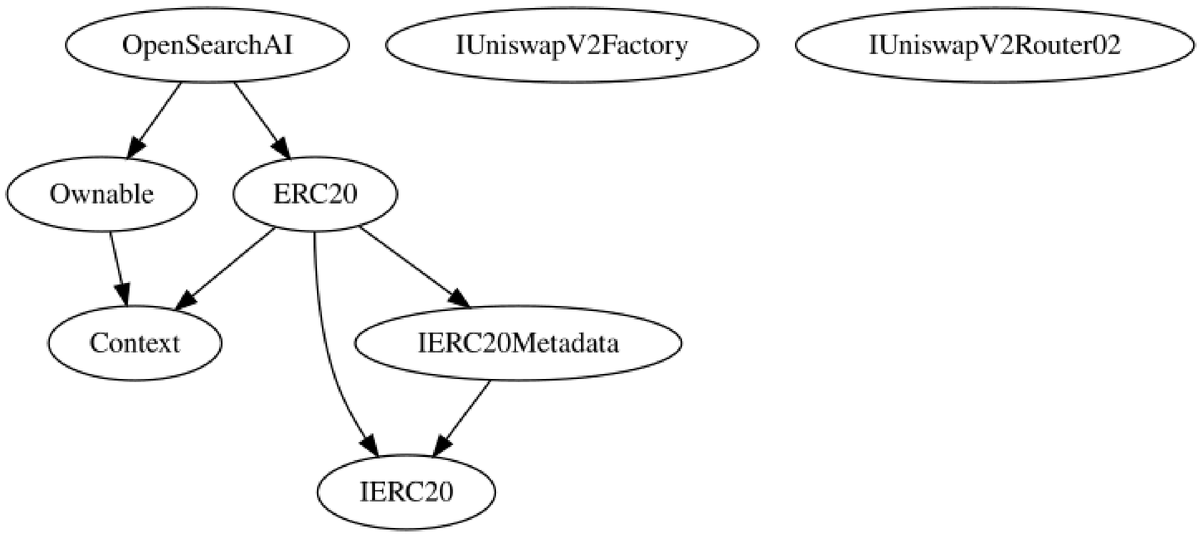
Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

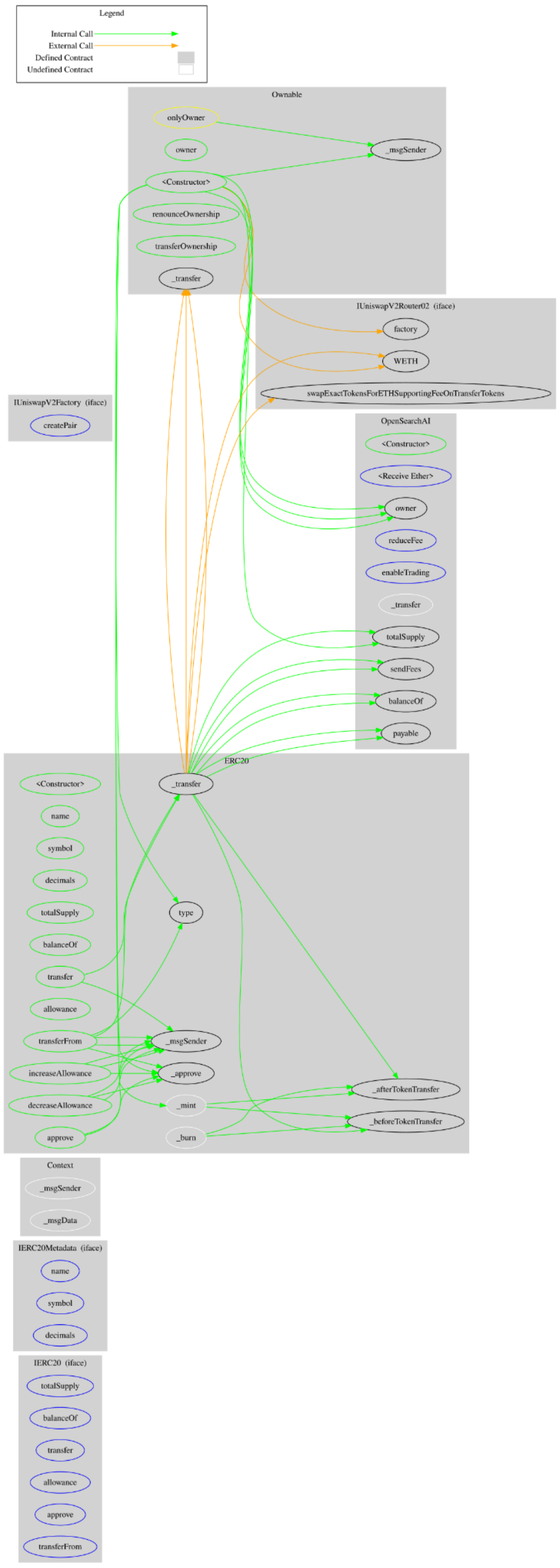
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
OpenSearchAI	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	sendFees	Internal	✓	
	reduceFee	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	_transfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
renounceOwnership
transferOwnership
_transfer
_msgSender
IUniswapV2Router02 (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
IUniswapV2Factory (iface)
createPair
OpenSearchAI
<Constructor>
<Receive Ether>
owner
reduceFee
enableTrading
_transfer
totalSupply
sendFees
balanceOf
payable
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_transfer
type
_msgSender
_approve
_mint
_burn
_afterTokenTransfer
_beforeTokenTransfer
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

OpenSearchAI contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. OpenSearchAI is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. The fees are set at 5% for buy and sell transactions.

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

https://etherscan.io/tx/0x463876c645a6f20ec9303885571c502259369ed48610b87c91592fdcc9a906bb

Disclaimer

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io