Cyberscope

Audit Report

AEZAI

February 2023

Type	ERC20
Network	ETH
Address	0x03929e58E4D1d02D071a940D0B2624347C9D0Ac8
Audited by	© cyberscope

Table of Contents

Review

Contract Name	AEZAI
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://etherscan.io/address/0x03929e58e4d1d02d071a940d0b2624347c9d0ac8
Address	0x03929e58e4d1d02d071a940d0b2624347c9d0ac8
Network	ETH
Symbol	AI
Decimals	18
Total Supply	1,000,000,000

Audit Updates

Initial Audit	07 Feb 2023

Source Files

Filename	SHA256
AEZAI.sol	579b0caac9f614117aee29bbcc7a13fca40d9577a505c79aa6d72a7544169e8b

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	AEZAI.sol#L214
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `changeMarketingFeePercent` function with a high percentage value.

```
function changeMarketingFeePercent(uint8 _tax) external onlyOwner
{
  marketingTax = _tax;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	AEZAI.sol#L181
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract performs redundant calculations.

```
uint256 tax = amount.mul(marketingTax).div(100);

amount = amount.sub(tax);
_balances[marketingWallet] = _balances[marketingWallet].add(tax);
emit Transfer(sender, marketingWallet, tax);
amount = amount.add(tax);

uint256 tax1 = amount.mul(marketingTax).div(100);
uint256 amount1 = amount.sub(tax1);
_balances[sender] = _balances[sender].sub(amount, "ERC20: transfer
amount exceeds balance");
_balances[recipient] = _balances[recipient].add(amount1);
emit Transfer(sender, recipient, amount);
```

Recommendation

The team is advised to take into consideration these segments and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

- It is recommended to remove redundant calculations.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	AEZAI.sol#L126,128,135,205,210,301
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
mapping (address => uint) internal _balances
mapping (address => mapping (address => uint)) internal _allowances
uint internal _totalSupply
address _wallet
uint8 _tax
address public _owner
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	AEZAI.sol#L131
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
address[] private _excluded
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	AEZAI.sol#L253,266,270,274,280
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 isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
...
function safeTransfer(IERC20 token, address to, uint value) internal {
        callOptionalReturn(token,
abi.encodeWithSelector(token.transfer.selector, to, value));
    }

function safeTransferFrom(IERC20 token, address from, address to, uint
value) internal {
        callOptionalReturn(token,
abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	AEZAI.sol#L207
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
marketingWallet = _wallet
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	AEZAI.sol#L257
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { codehash := extcodehash(account) }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	AEZAI.sol#L11
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.17;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

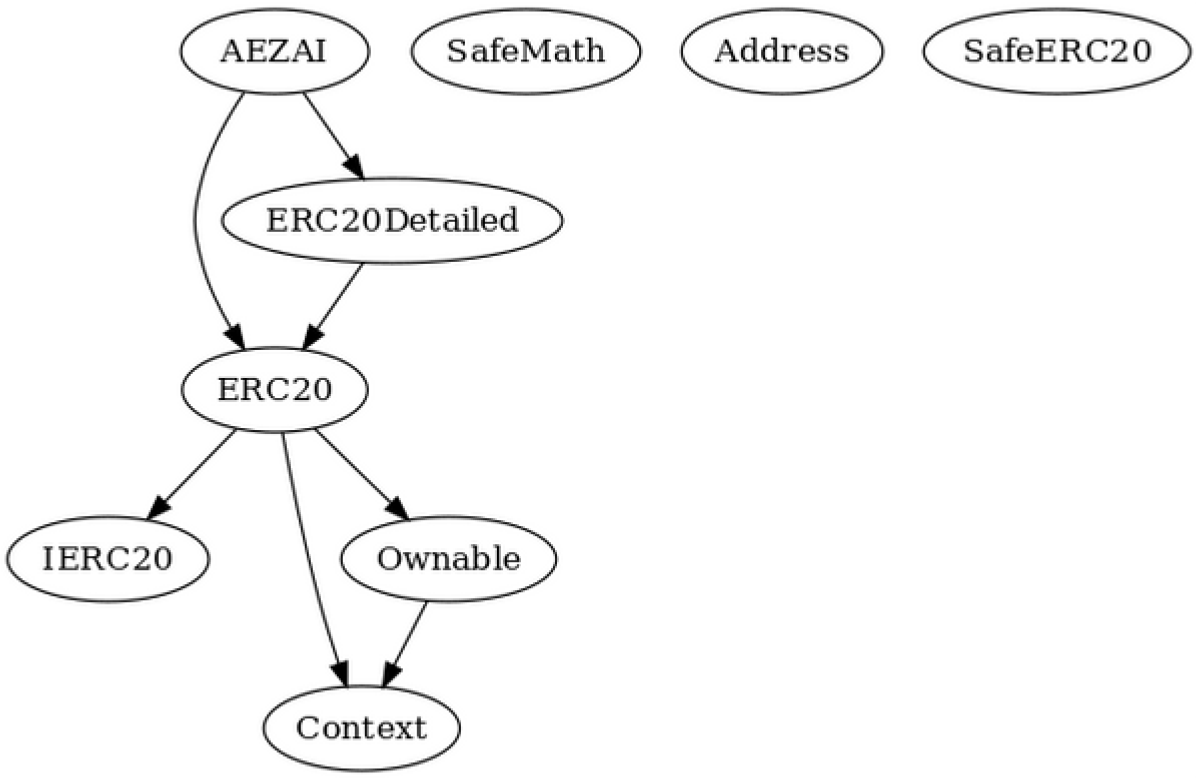
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
Context	Implementation			
		Public	✓	-
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner

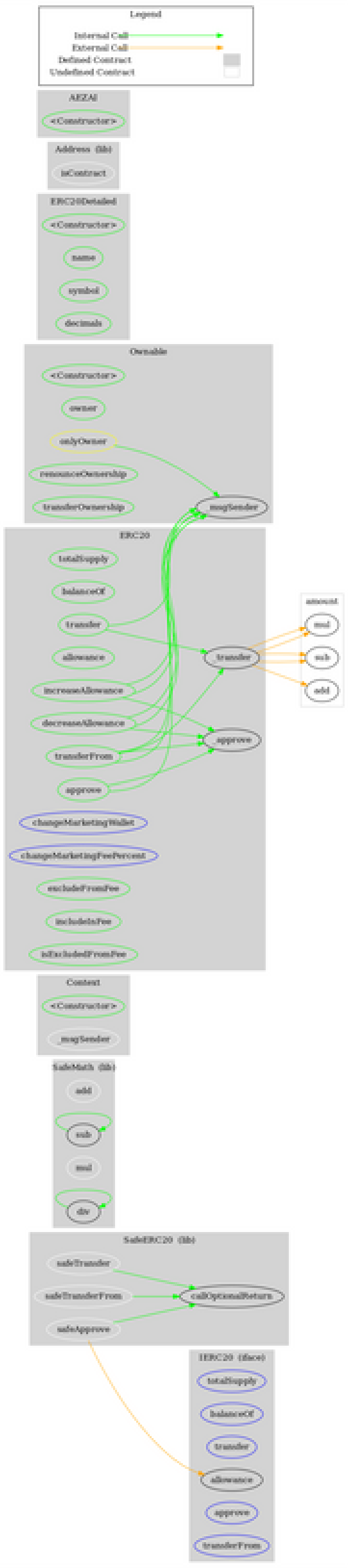
	transferOwnership	Public	✓	onlyOwner
ERC20	Implementation	Context, Ownable, IERC20		
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_approve	Internal	✓	
	changeMarketingWallet	External	✓	onlyOwner
	changeMarketingFeePercent	External	✓	onlyOwner
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	isExcludedFromFee	Public		-
ERC20Detailed	Implementation	ERC20		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
Address	Library			
	isContract	Internal		

SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	callOptionalReturn	Private	✓	
AEZAI	Implementation	ERC20, ERC20Detailed		
		Public	✓	ERC20Detailed

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
AEZAI
<Constructor>
Address (lib)
isContract
ERC20Detailed
<Constructor>
name
symbol
decimals
Ownable
<Constructor>
owner
onlyOwner
renounceOwnership
transferOwnership
msgSender
ERC20
totalSupply
balanceOf
transfer
allowance
increaseAllowance
decreaseAllowance
transferFrom
approve
changeMarketingWallet
changeMarketingFeePercent
excludedFromFee
includeInFee
isExcludedFromFee
_transfer
_approve
amount
mul
sub
add
Context
<Constructor>
msgSender
SafeMath (lib)
add
sub
mul
div
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
callOptionalReturn
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

There are some functions that can be abused by the owner like manipulating the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

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