



Cyberscope

Audit Report

Football AI Token

March 2023

Type BEP20

Network BSC

Address 0x4CDB15a578B5635B5D5CdC16f62f27e05C765C0b

Audited by © cyberscope

Table of Contents

Table of Contents	1
Review	1
Audit Updates	2
Source Files	2
Analysis	2
ST - Stops Transactions	3
Description	3
Recommendation	4
ELFM - Exceeds Fees Limit	4
Description	4
Recommendation	5
Diagnostics	5
RSML - Redundant SafeMath Library	6
Description	6
Recommendation	7
L02 - State Variables could be Declared Constant	7
Description	8
Recommendation	8
L04 - Conformance to Solidity Naming Conventions	8
Description	9
Recommendation	9
L07 - Missing Events Arithmetic	10
Description	10
Recommendation	11
L09 - Dead Code Elimination	11
Description	12
Recommendation	12
L17 - Usage of Solidity Assembly	13
Description	13
Recommendation	14
L19 - Stable Compiler Version	14
Description	14
Recommendation	15
Functions Analysis	15
Inheritance Graph	21
Flow Graph	21
Summary	21

Disclaimer**24****About Cyberscope****25**

Review

Contract Name	FootballAI
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://bscscan.com/address/0x4cdb15a578b5635b5d5cdc16f62f27e05c765c0b
Address	0x4cdb15a578b5635b5d5cdc16f62f27e05c765c0b
Network	BSC
Symbol	FAT
Decimals	18
Total Supply	10.000.000.000.000.000

Audit Updates

Initial Audit	14 Mar 2023
---------------	-------------

Source Files

Filename	SHA256
FootballAI.sol	a4a09f7db16c7f47b0bfe89c251b8ef0fc50365d7a3e24e01fb9b09721bf75ad

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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

ST - Stops Transactions

Criticality	Medium
Location	FootballAI.sol#L646
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` to zero.

```
function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**3
    );
}
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	FootballAI.sol#L527,530,533
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 `setTaxFeePercent`, `setDevelopmentFeePercent` and `setLiquidityFeePercent` functions with a high percentage value.

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    _taxFee = taxFee;
}
function setDevelopmentFeePercent(uint256 developmentFee) external
onlyOwner() {
    _developmentFee = developmentFee;
}
function setLiquidityFeePercent(uint256 liquidityFee) external
onlyOwner() {
    _liquidityFee = liquidityFee;
}
```

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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	FootballAI.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases unnecessarily the gas consumption.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on

<https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes>.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	FootballAI.sol#L374,376,379,380,381,394
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.

```
address private _developmentWalletAddress =
0xC92D3B06729eFbaE6B042188680DE3b22F384C5a
uint256 private _tTotal = 100000000000000 * 10**20
string private _name = "Football AI"
string private _symbol = "FAT"
uint8 private _decimals = 18
uint256 private numTokensSellToAddToLiquidity = 100000000000000 *
10**20
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	FootballAI.sol#L202,203,217,235,382,384,386,393,537,595,600,605
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.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
uint256 public _taxFee = 0
uint256 public _developmentFee = 0
uint256 public _liquidityFee = 10
uint256 public _maxTxAmount = 100000000000000 * 10**20
bool _enabled
uint256 _amount
```

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>.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	FootballAI.sol#L524,527,530,533
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_taxFee = taxFee
_developmentFee = developmentFee
_liquidityFee = liquidityFee

_maxTxAmount = _tTotal.mul(maxTxPercent).div(
    10**3
)
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	FootballAI.sol#L96,101,106,109,112,115,121,124,129,132,137
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) {
    uint256 size;
    assembly { size := extcodesize(account) }
    return size > 0;
}

...

(bool success, ) = recipient.call{ value: amount }("");
require(success, "Address: unable to send value, recipient may
have reverted");
}

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

...
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	FootballAI.sol#L98,142
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 { size := extcodesize(account) }

assembly {
    let returndata_size := mload(returndata)
    revert(add(32, returndata), returndata_size)
}
```

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	FootballAI.sol#L3
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 ^0.8.4;
```

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			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Context	Implementation			

	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-

	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-

	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-

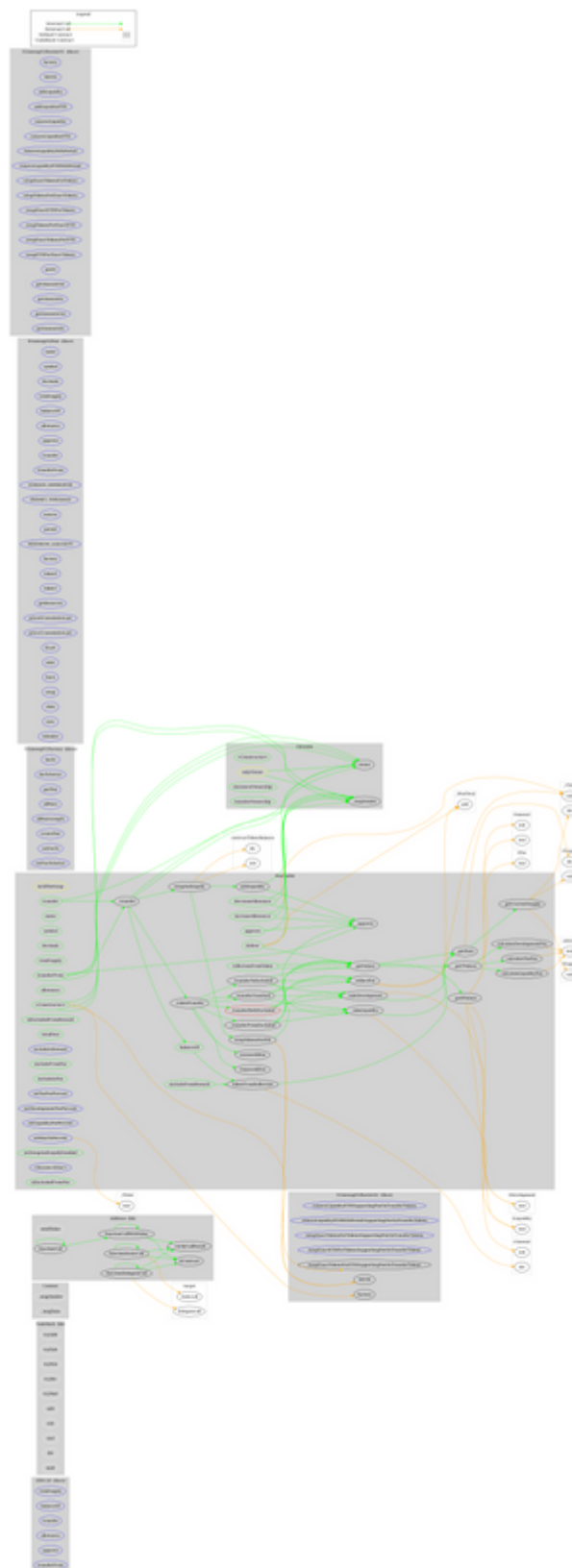
	swapExactETHForTokensSupporting FeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupporting FeeOnTransferTokens	External	✓	-
FootballAI	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	deliver	Public	✓	-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setTaxFeePercent	External	✓	onlyOwner
	setDevelopmentFeePercent	External	✓	onlyOwner

	setLiquidityFeePercent	External	✓	onlyOwner
	setMaxTxPercent	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	_takeDevelopment	Private	✓	
	calculateTaxFee	Private		
	calculateDevelopmentFee	Private		
	calculateLiquidityFee	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-
	_approve	Private	✓	
	_transfer	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	

Inheritance Graph



Flow Graph



Summary

Football AI Token contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and manipulate 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>