Cyberscope

Audit Report

Fashion AI

February 2023

Type	BEP20
Network	BSC
Address	0xd9a6d7b177c5f963501b69f0ed08a49e3a6f71ee
Audited by	© cyberscope

Table of Contents

Review

Contract Name	FashionAIToken
Compiler Version	v0.8.2+commit.661d1103
Optimization	200 runs
Explorer	https://etherscan.io/address/0xd9a6d7b177c5f963501b69f0ed08a49e3a6f71ee
Address	0xd9a6d7b177c5f963501b69f0ed08a49e3a6f71ee
Network	ETH
Symbol	FAI
Decimals	18
Total Supply	500,000,000

Audit Updates

Initial Audit	18 Feb 2023

Source Files

Filename	SHA256
contracts/FashionAI.sol	f0479c0b772045bda6a21dabe29a7673c61eb782c1c0d6d5312b32cae27b0aca

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	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PTRP	Potential Transfer Revert Propagation	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
●	L14	Uninitialized Variables in Local Scope	Unresolved

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/FashionAI.sol#L756
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
bool sent = payable(marketingWallet).send(
    address(this).balance
);
require(sent, "Failed to send ETH");
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be archived by not allowing set contract addresses or by sending the funds in a non-revertable way.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/FashionAI.sol#L628,629,630,631,640,642,643,644,645,646,647
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.

```
string private _name = "Fashion AI"
string private _symbol = "FAI"
uint8 private _decimals = 18
uint256 private _supply = 500000000
address public DEAD = 0x000000000000000000000000000000000000dEaD
address public presaleWallet = 0x7c57dd1C63Ebb5D99a28a8932c36fbCA036fc74c
address public liquidStakingWallet =
0x2D58D4032eC78e485E6f394e0950e018eafDE9E0
address public dappFashionAIWallet =
0x4Cc7a7F9900b18712158B0d0a35C1CE244E29b7c
address public firstBurnWallet  = 0xE746277A8D89dB1223753B8C4192d4a2b952E9e6
address public cexListingWallet = 0xa867081b7EC09B243B55C806E418781c5f375C89
address public airdropWallet = 0x613a44e38f44e8C690FFD5E4eaCB068488D66c1f
```

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	contracts/FashionAI.sol#L38,640,652,653,708,791,855,856,869,870
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 WETH() external pure returns (address);
address public DEAD = 0x000000000000000000000000000000000000dEaD
uint256 public _marketingReserves = 0
mapping(address => bool) public _isExcludedFromFee
address _pair
bool _status
address _address
uint256 _buyTaxForLiquidity
uint256 _sellTaxForMarketing
uint256 _numTokensSellToAddToLiquidity
uint256 _numTokensSellToAddToETH
```

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	contracts/FashionAI.sol#L862,880
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.

```
buyTaxForLiquidity = _buyTaxForLiquidity

numTokensSellToAddToLiquidity =
            _numTokensSellToAddToLiquidity *
            10 ** _decimals
```

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	contracts/FashionAI.sol#L512
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");

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds
balance");
        unchecked {
            _balances[account] = accountBalance - amount;
            // Overflow not possible: amount <= accountBalance <= totalSupply.
            _totalSupply -= amount;
        }

        emit Transfer(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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/FashionAI.sol#L764,765
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 marketingShare
uint256 liquidityShare
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

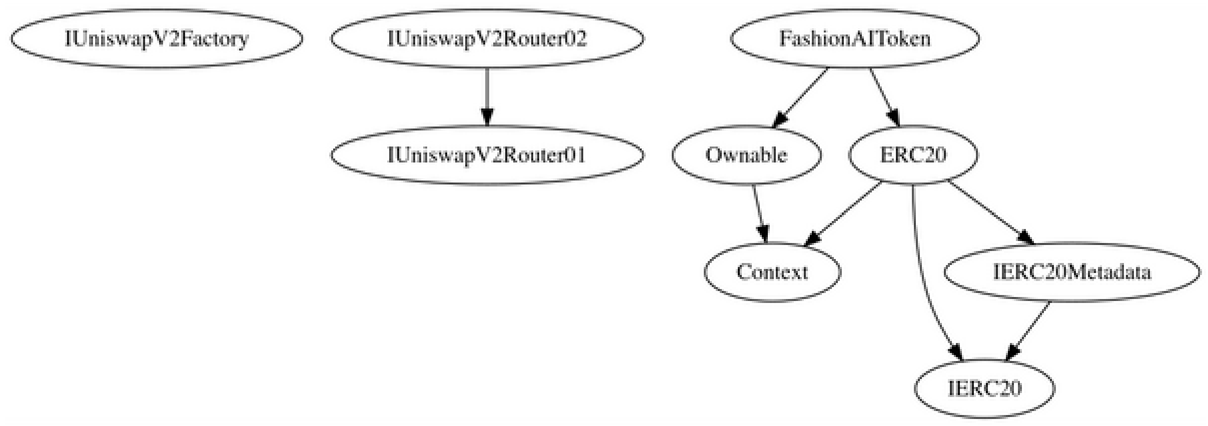
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	decimals	External		-

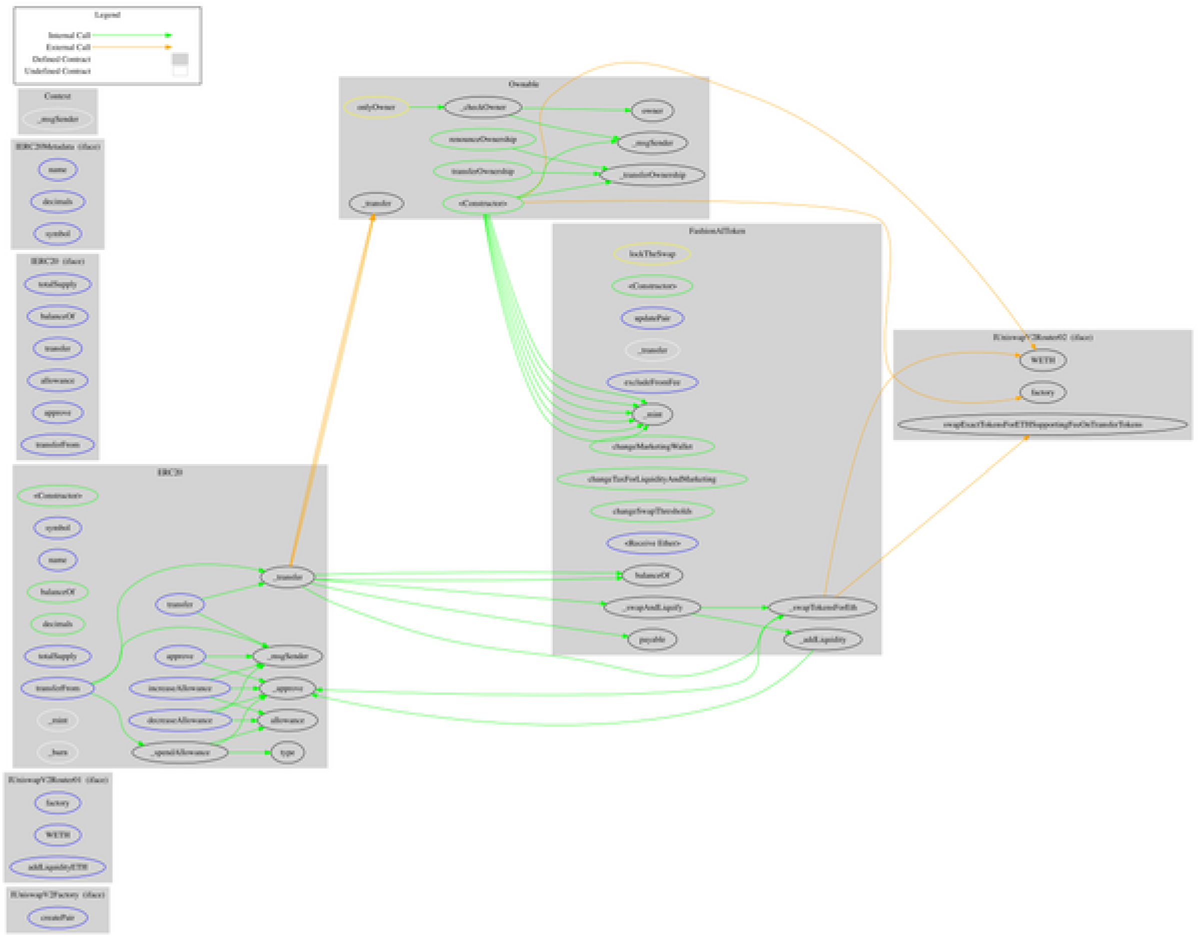
	symbol	External		-
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	symbol	External		-
	name	External		-
	balanceOf	Public		-
	decimals	Public		-
	totalSupply	External		-
	allowance	Public		-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
	decreaseAllowance	External	✓	-
	increaseAllowance	External	✓	-
	_mint	Internal	✓	
	_burn	Internal	✓	

	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_transfer	Internal	✓	
FashionAIToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	updatePair	External	✓	onlyOwner
	_transfer	Internal	✓	
	excludeFromFee	External	✓	onlyOwner
	_swapAndLiquify	Private	✓	lockTheSwap
	_swapTokensForEth	Private	✓	lockTheSwap
	_addLiquidity	Private	✓	lockTheSwap
	changeMarketingWallet	Public	✓	onlyOwner
	changeTaxForLiquidityAndMarketing	Public	✓	onlyOwner
	changeSwapThresholds	Public	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

Fashion AI 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. There is also a limit of max 10% fees.

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