MIXED CRYPTO ARTS

Smart Contract Audit

Terrance Nibbles - Certified Auditor

April 27, 2025

MIXED CRYPTO ARTS

Smart Contract Audit

Preface

This audit is of the MIXED CRYPTO ARTS Token contract that was provided for detailed analysis on April 25, 2025. The primary solidity smart contract code is listed at the end of the report. This was manually audited as well as reviewed with other tools.

This token contract that was audited is on the ETHEREUM Blockchain:

https://etherscan.io/token/0xFCC63eCc963A75e46aAf6962Aaaf1e18B7FD7e59#code

DISCLAIMER:

This audit report is based on a professional review of the provided smart contract provided. It is important to note that this assessment represents our expert opinion and analysis of the code at the time of the evaluation. The findings and recommendations presented herein are not intended to serve as warranties, guarantees, or assurances of the contract's performance, security, or functionality on any live network, including the Ethereum mainnet.

We expressly disclaim any responsibility for errors, omissions, or inaccuracies in this report, as the assessment is conducted on a non-exhaustive basis and may not cover all possible scenarios or future developments. The audit is conducted in accordance with industry best practices and standards at the time of evaluation.

Furthermore, we are unable to confirm the deployment of this specific contract on the Ethereum mainnet. This report is solely based on the provided code and does not verify the actual deployment status on any live blockchain. It is the responsibility of the contract deployer to ensure the accurate deployment of the contract and adhere to security best practices when deploying to production environments.

Users, developers, and stakeholders are advised to perform additional due diligence and testing before deploying or interacting with the contract on any live network. This report should be considered as a tool for risk assessment rather than a guarantee of the contract's security or performance. In the dynamic and rapidly evolving field of blockchain technology, risks and vulnerabilities may emerge over time, and it is crucial to stay vigilant and up-to-date on security best practices.

By relying on this audit report, the reader acknowledges and accepts that the audit is based on the provided information and that no warranties, guarantees, or assurances are expressed or implied.

🧠 Security Audit Report: Mixed Crypto Arts (MCA)

Contract Address: 0xFCC63eCc963A75e46aAf6962Aaaf1e18B7FD7e59
Language: Solidity 0.8.26
Frameworks: OpenZeppelin (Ownable, ERC20)
Tax Mechanics, Uniswap Integration, Custom Wallet Limit Logic

🧱 Key Components

Feature	Present	Notes
ERC-20 Standard	✅	Built on OpenZeppelin
Ownership Control	✅	via `Ownable`
Max Wallet Size	✅	Anti-whale feature
Transfer Tax	✅	Redirects % to tax wallet
Fee Whitelist	✅	Owner and tax wallet
Uniswap Pair Creation	✅	At constructor
Burnable	✅	Owner-only

✅ Contract Overview

- **Token Name:** Mixed Crypto Arts
- **Symbol:** MCA
- **Total Supply:** `888,888,888,888,888` MCA
- **Tax System:** 4% by default, adjustable up to 25%
- **Max Wallet Limit:** Set to `44,444,444,444,440` tokens
- **Router:** Uses UniswapV2 Router02
- **Tax Wallet:** Initially `0x426f...e0a`

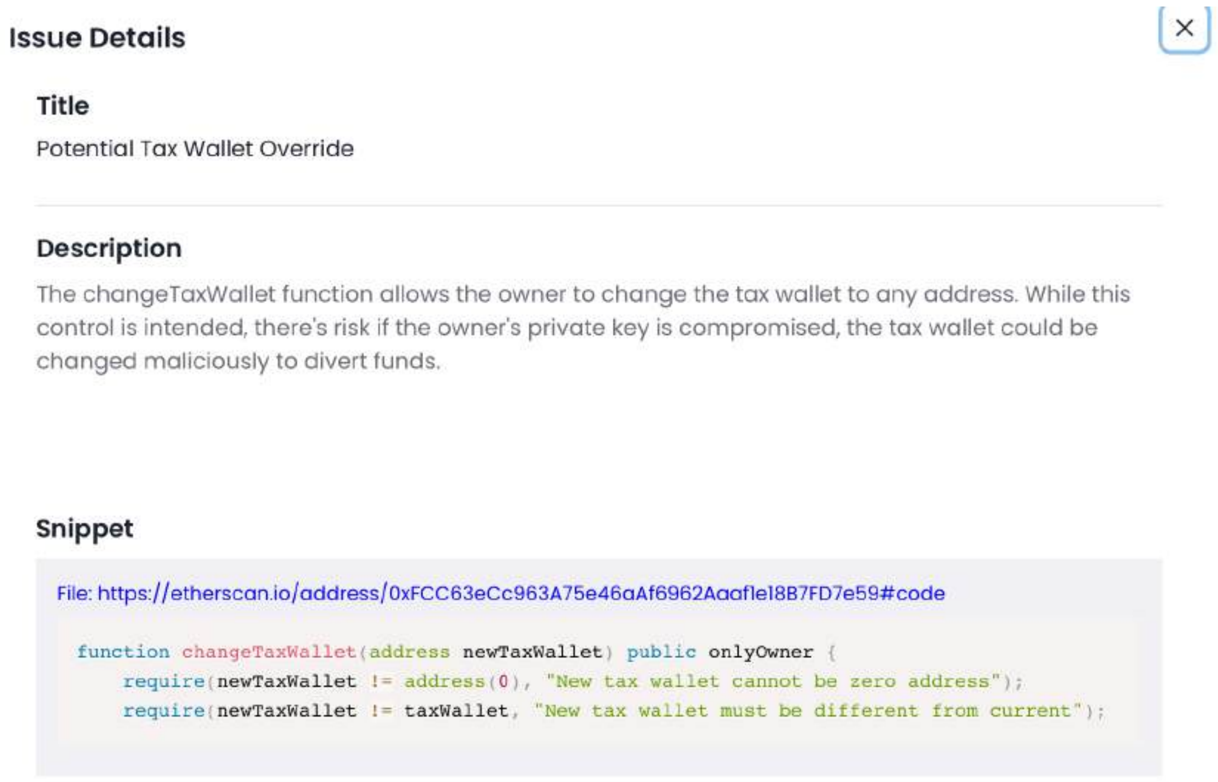
🔐 Security Review

✔️ Core ERC20 Functions

- Implements ERC20 using OpenZeppelin standards — ✅ Safe
- Uses OpenZeppelin's `Ownable` contract — ✅ Safe
- Tax mechanism is integrated into `_transfer` with conditional whitelist — ✅ Correctly scoped

⚠️ Tax Logic

- Tax is applied unless sender or recipient is whitelisted.
- Tax is hardcoded to be sent to a centralized `taxWallet`.
- **Risk:** Owner can change `taxWallet` to any address, potential security risk if ownership is compromised.



Issue Details

Title

Potential Tax Wallet Override

Description

The changeTaxWallet function allows the owner to change the tax wallet to any address. While this control is intended, there's risk if the owner's private key is compromised, the tax wallet could be changed maliciously to divert funds.

Snippet

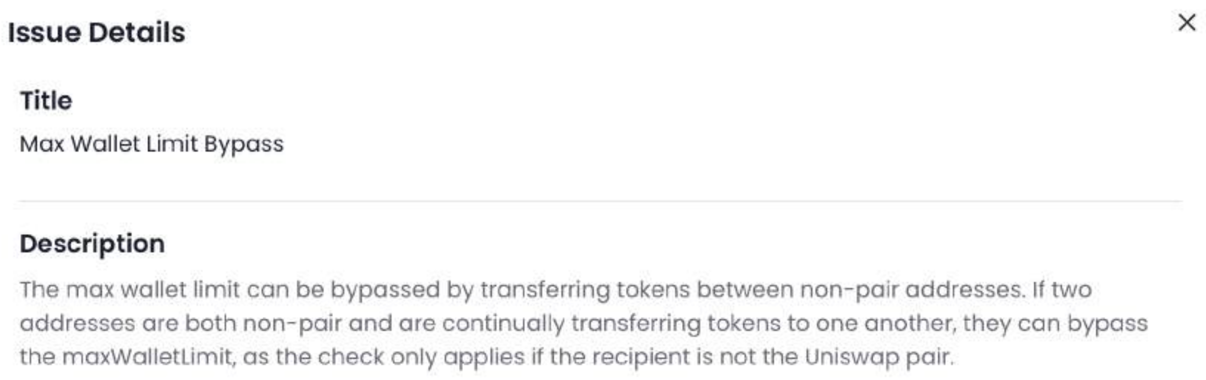
File: https://etherscan.io/address/0xFCC63eCc963A75e46aAf6962Aaafle18B7FD7e59#code

```
function changeTaxWallet(address newTaxWallet) public onlyOwner {
    require(newTaxWallet != address(0), "New tax wallet cannot be zero address");
    require(newTaxWallet != taxWallet, "New tax wallet must be different from current");
```

    - ✅ However, contract prevents `taxWallet` from being set to `0x0` or to the current wallet.

⚠️ Max Wallet Limit

- The `maxWalletLimit` is enforced **only on inbound transfers (excluding from LP)**.
- **Risk:** Does not protect against circumvention via direct mint or contract interactions.



Issue Details

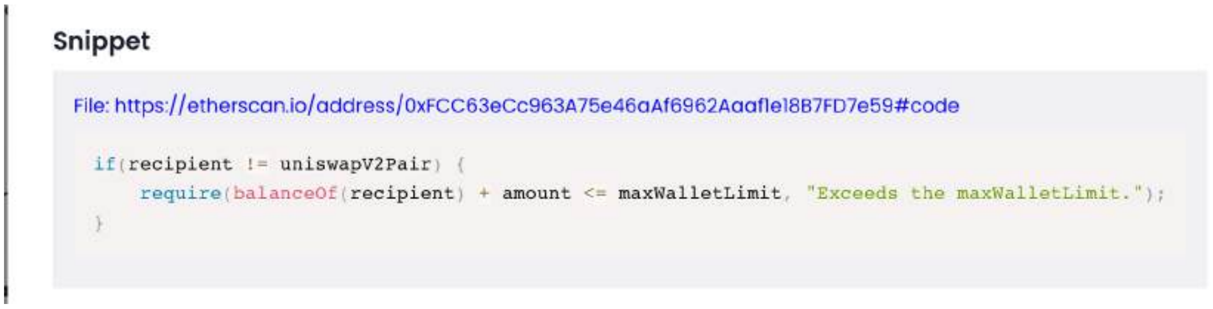
×

Title

Max Wallet Limit Bypass

Description

The max wallet limit can be bypassed by transferring tokens between non-pair addresses. If two addresses are both non-pair and are continually transferring tokens to one another, they can bypass the maxWalletLimit, as the check only applies if the recipient is not the Uniswap pair.



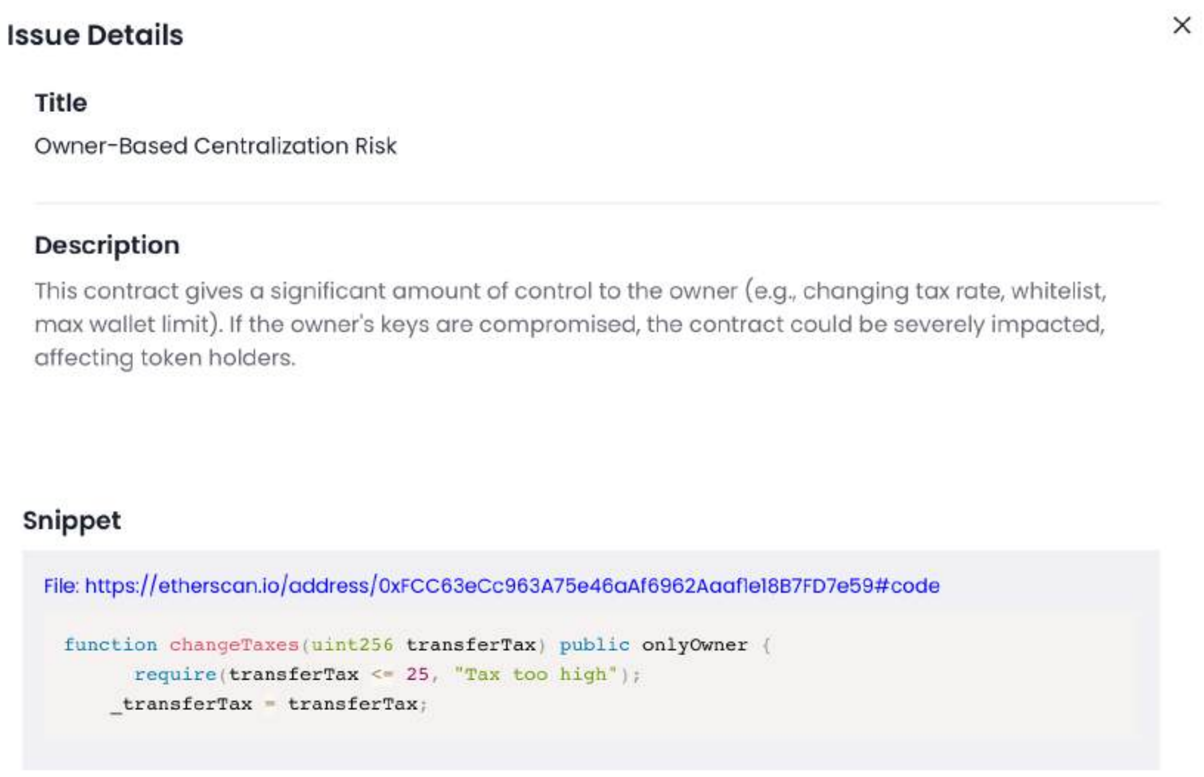
Snippet

File: https://etherscan.io/address/0xFCC63eCc963A75e46aAf6962Aaafle18B7FD7e59#code

```
if(recipient != uniswapV2Pair) {
    require(balanceOf(recipient) + amount <= maxWalletLimit, "Exceeds the maxWalletLimit.");
}
```

⚠️ Whitelisting

- Functions allow any address to be included or excluded from tax via `onlyOwner`.
- **Risk:** Centralized trust required in owner.



Issue Details

×

Title

Owner-Based Centralization Risk

Description

This contract gives a significant amount of control to the owner (e.g., changing tax rate, whitelist, max wallet limit). If the owner's keys are compromised, the contract could be severely impacted, affecting token holders.

Snippet

File: https://etherscan.io/address/0xFCC63eCc963A75e46aAf6962Aaafle18B7FD7e59#code

```
function changeTaxes(uint256 transferTax) public onlyOwner {
      require(transferTax <= 25, "Tax too high");
    _transferTax = transferTax;
```

✅ No Overflows

- Uses `SafeMath`, although unnecessary since Solidity ^0.8.0 has built-in overflow checks.

🔍 Security & Design Issues

🔶 Transfer Logic

```
solidity
if(!_isWhiteListedFromFee[sender] && !
_isWhiteListedFromFee[recipient]){
    if(recipient != uniswapV2Pair){
        require(balanceOf(recipient) + amount <=
maxWalletLimit, "Exceeds the maxWalletLimit.");
    }
    if(_transferTax > 0) {
        uint256 fee = amount.mul(_transferTax).div(100);
        transferAmount = amount.sub(fee);
        super._transfer(sender, taxWallet, fee);
    }
}
```

🔒 **Access Control:** Only applies to non-whitelisted senders and receivers. Properly excludes owner and tax wallet.

🚨 Risk 1: Fee Bypass via Pair Transfer

- Transfers to the Uniswap pair **bypass the max wallet restriction**.
- This is intentional for LP functionality but could be abused to sidestep restrictions.

🚨 Risk 2: Tax Wallet is Fully Trusted

- The tax wallet gets direct fees. If compromised, it drains a % of every transaction.

💡 **Improvement:** Add time-lock or multi-sig pattern to `changeTaxWallet()`.

🔶 Tax Modification

```
solidity
function changeTaxes(uint256 transferTax) public onlyOwner
{
    require(transferTax <= 25, "Tax too high");
    _transferTax = transferTax;
}
```

✅ Max tax cap (25%) prevents hard rug-pull vector.

⚠️ **Owner can still change this at will** — though limited, it's still centralized control.

🔶 Whitelist Functions

```
solidity
function whiteListFromFee(address account) public onlyOwner
function includeInFee(address account) public onlyOwner
```

✅ Clear, toggles fee exemptions. 🚨 **Risk:** Owner could silently exempt bots or malicious contracts from tax + wallet limit.

💡 Suggest tracking whitelist changes via events for transparency.

🔶 Max Wallet Limit

```
solidity
require(_limit > totalSupply().div(200),"Limit too low");
```

✅ Enforces a minimum of 0.5% total supply. Sensible anti-whale mechanism.

🚨 If users accumulate via external transfers (like airdrops), this limit might be unintentionally exceeded, leading to blocked trades.

🔶 burnTokens

```
solidity
function burnTokens(uint256 amount) public onlyOwner
```

🚨 **Centralization Risk:** Owner can deflate supply — generally positive, but if misused can spike token price.

🔶 Uniswap Setup

```
solidity
_uniswapV2Router =
IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F248
8D);
uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory()).createPair(ad
dress(this), _uniswapV2Router.WETH());
```

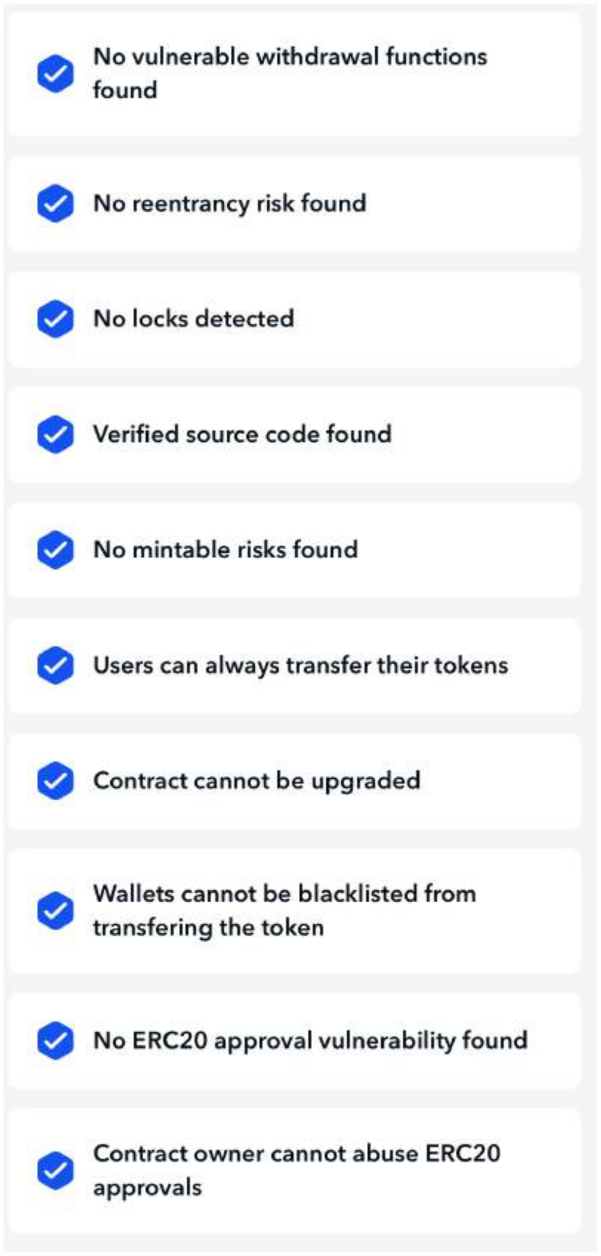
✅ Correct mainnet router and WETH pairing. 🚨 No sanity check to ensure router is valid.

💡 Consider adding `require(uniswapV2Pair != address(0))` for fail-safe.

🧾 Final Verdict

Category	Status	Details
Access Control	✅ Safe	`onlyOwner` used correctly
Upgradeability	❌ None	Non-upgradable
Tokenomics Transparency	⚠️ Medium	Centralized tax control
External Calls	✅ Safe	No direct user-facing `call` usage
Reentrancy	✅ Not applicable	No payable logic
Mint/Burn Control	✅ Safe	Only `_mint` in constructor
Max Wallet Logic	✅ Strong	With smart thresholds
Fee Logic	⚠️ Moderate	Owner can disable or redirect all fees
Overall Risk Level	🟢 Low	Minor issues, mostly centralized control concerns

NO HIGH RISK ISSUES IDENTIFIED


✓ No vulnerable withdrawal functions found

✓ No reentrancy risk found

✓ No locks detected

✓ Verified source code found

✓ No mintable risks found

✓ Users can always transfer their tokens

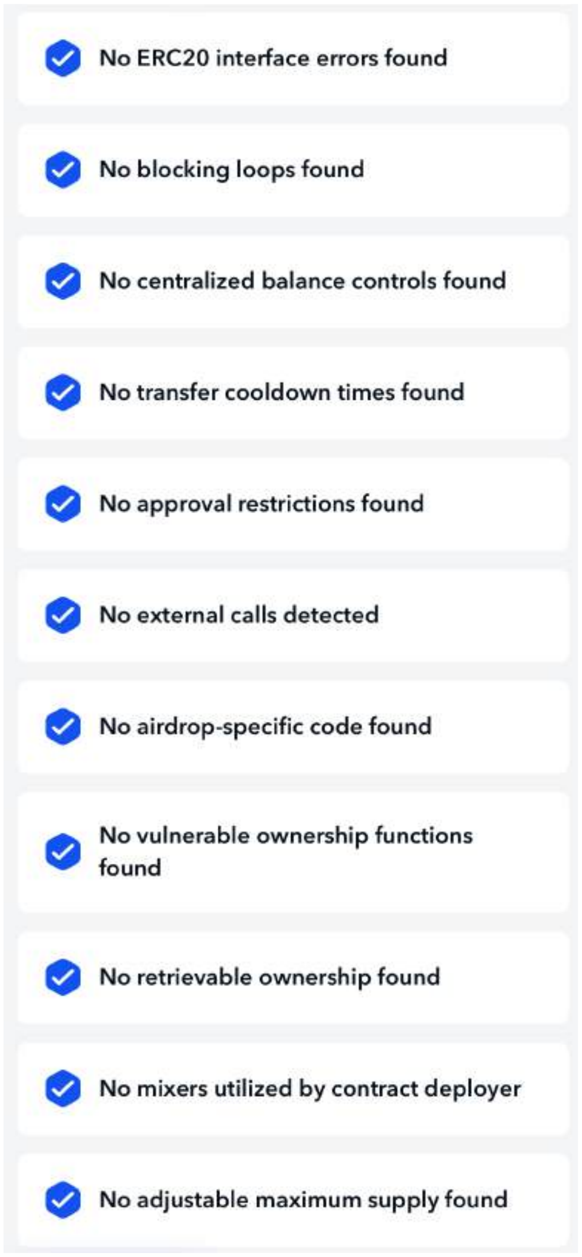
✓ Contract cannot be upgraded

✓ Wallets cannot be blacklisted from transfering the token

✓ No ERC20 approval vulnerability found

✓ Contract owner cannot abuse ERC20 approvals

```
 // SPDX-License-Identifier: MIT
/**
 *Submitted for verification at Etherscan.io on 2025-04-18
*/

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)

/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}

// File @openzeppelin/contracts/access/Ownable.sol@v4.8.1

// OpenZeppelin Contracts (last updated v4.7.0) (access/Ownable.sol)

/**
 * @dev Contract module which provides a basic access control mechanism,
where
 * there is an account (an owner) that can be granted exclusive access to
```



✓ No ERC20 interface errors found

✓ No blocking loops found

✓ No centralized balance controls found

✓ No transfer cooldown times found

✓ No approval restrictions found

✓ No external calls detected

✓ No airdrop-specific code found

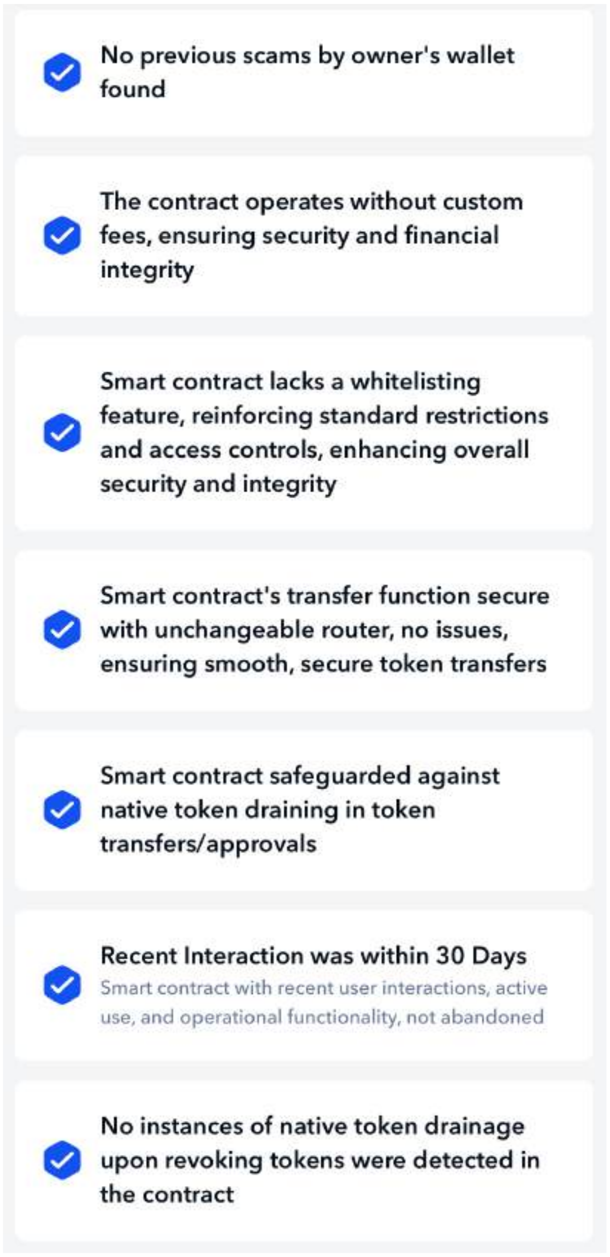
✓ No vulnerable ownership functions found

✓ No retrievable ownership found

✓ No mixers utilized by contract deployer

✓ No adjustable maximum supply found

```
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(
        address indexed previousOwner,
        address indexed newOwner
    );

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _transferOwnership(_msgSender());
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        _checkOwner();
        _;
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }
```



✓ No previous scams by owner's wallet found

✓ The contract operates without custom fees, ensuring security and financial integrity

✓ Smart contract lacks a whitelisting feature, reinforcing standard restrictions and access controls, enhancing overall security and integrity

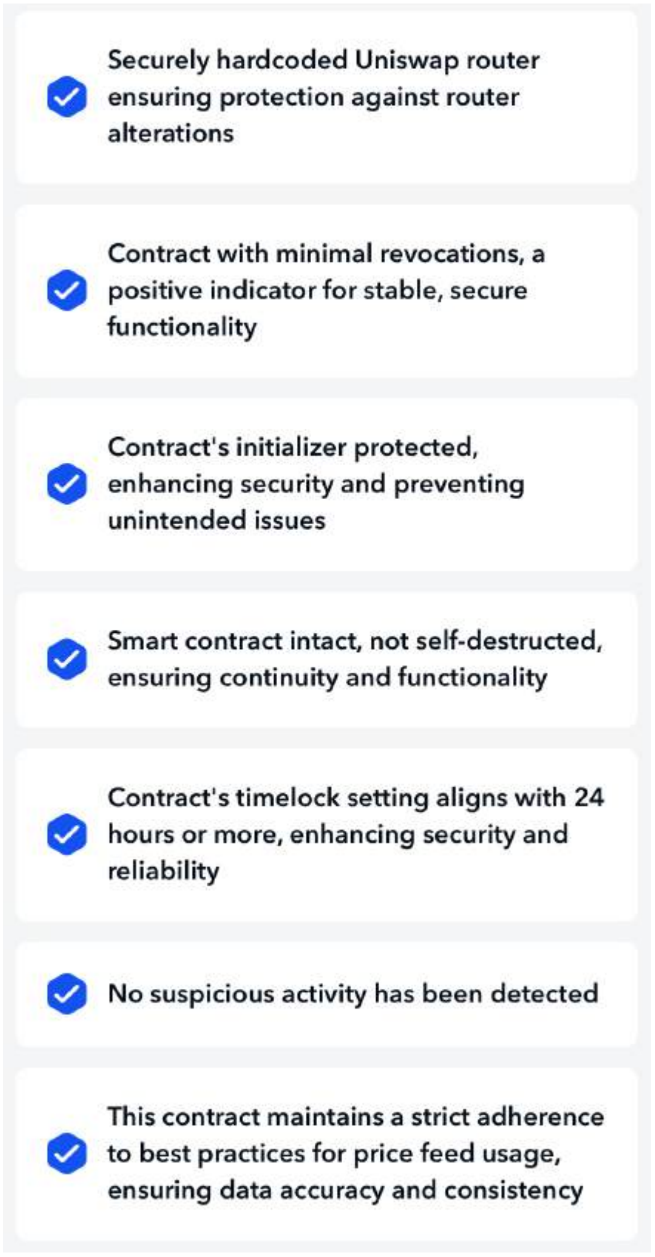
✓ Smart contract's transfer function secure with unchangeable router, no issues, ensuring smooth, secure token transfers

✓ Smart contract safeguarded against native token draining in token transfers/approvals

✓ Recent Interaction was within 30 Days
Smart contract with recent user interactions, active use, and operational functionality, not abandoned

✓ No instances of native token drainage upon revoking tokens were detected in the contract

```
/**
 * @dev Throws if the sender is not the owner.
 */
function _checkOwner() internal view virtual {
   require(owner() == _msgSender(), "Ownable: caller is not the owner");
}

/**
 * @dev Leaves the contract without owner. It will not be possible to call
 * `onlyOwner` functions anymore. Can only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave the contract without an owner,
 * thereby removing any functionality that is only available to the owner.
 */
function renounceOwnership() public virtual onlyOwner {
   _transferOwnership(address(0));
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public virtual onlyOwner {
   require(
      newOwner != address(0),
      "Ownable: new owner is the zero address"
   );
   _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Internal function without access restriction.
 */
function _transferOwnership(address newOwner) internal virtual {
   address oldOwner = _owner;
   _owner = newOwner;
  emit OwnershipTransferred(oldOwner, newOwner);
```



✓ Securely hardcoded Uniswap router ensuring protection against router alterations

✓ Contract with minimal revocations, a positive indicator for stable, secure functionality

✓ Contract's initializer protected, enhancing security and preventing unintended issues

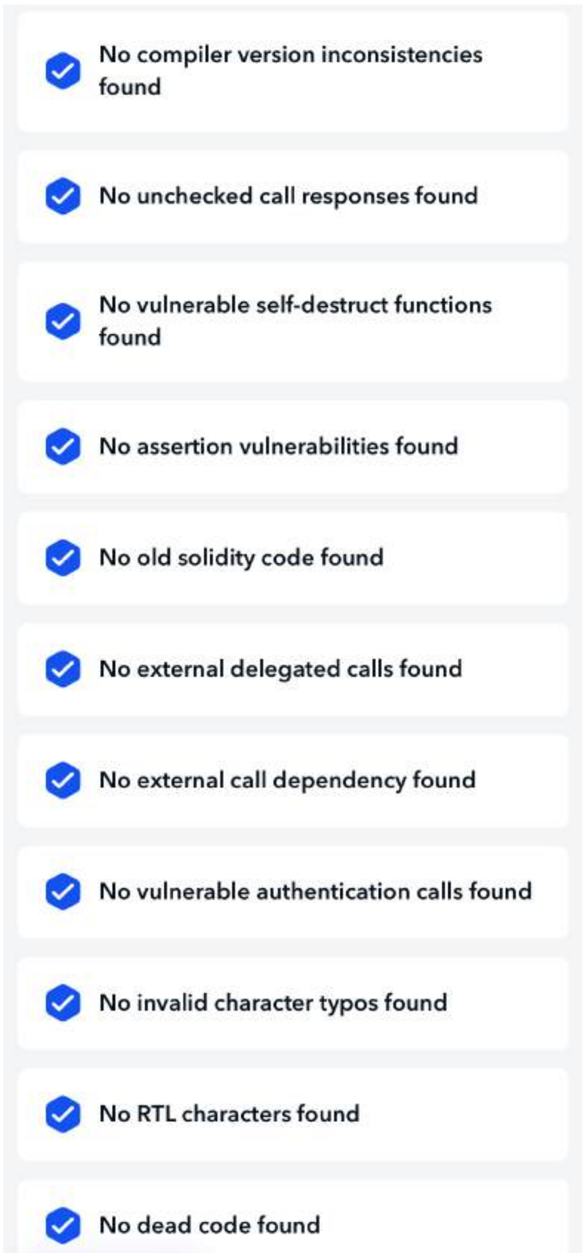
✓ Smart contract intact, not self-destructed, ensuring continuity and functionality

✓ Contract's timelock setting aligns with 24 hours or more, enhancing security and reliability

✓ No suspicious activity has been detected

✓ This contract maintains a strict adherence to best practices for price feed usage, ensuring data accuracy and consistency

```
    }
}

// File @openzeppelin/contracts/token/ERC20/IERC20.sol@v4.8.1
// OpenZeppelin Contracts (last updated v4.6.0) (token/ERC20/IERC20.sol)

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Emitted when `value` tokens are moved from one account
(`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set
by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );

    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
```



✓ No compiler version inconsistencies found

✓ No unchecked call responses found

✓ No vulnerable self-destruct functions found

✓ No assertion vulnerabilities found

✓ No old solidity code found

✓ No external delegated calls found

✓ No external call dependency found

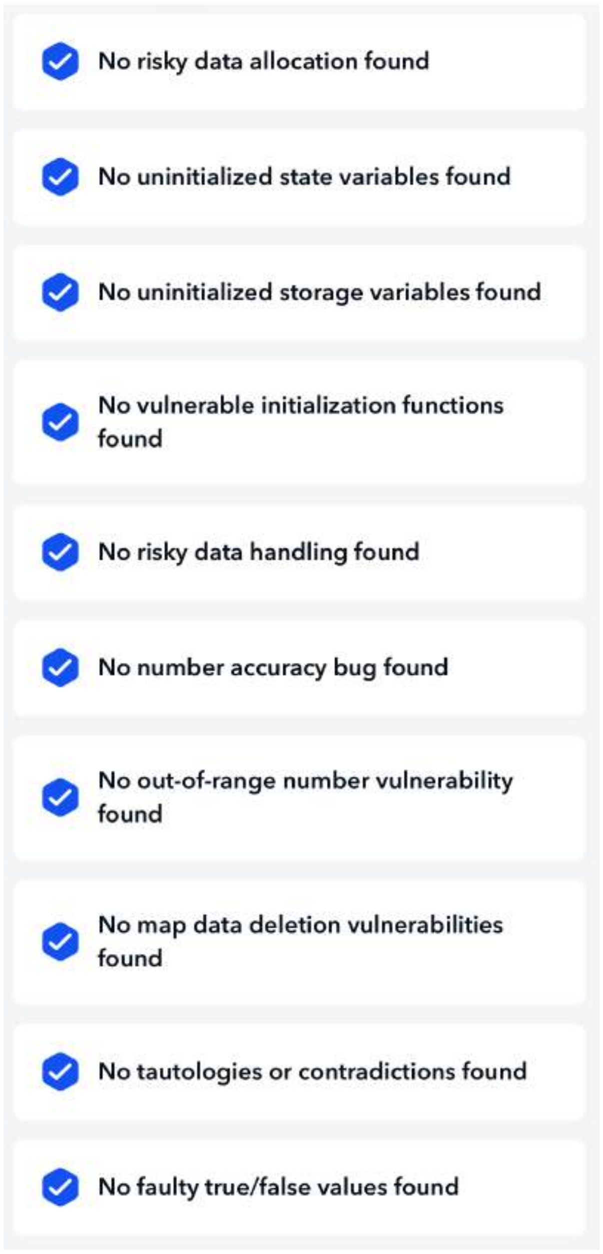
✓ No vulnerable authentication calls found

✓ No invalid character typos found

✓ No RTL characters found

✓ No dead code found

```
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `to`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address to, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(
        address owner,
        address spender
    ) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the
risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
```


✓ No risky data allocation found

✓ No uninitialized state variables found

✓ No uninitialized storage variables found

✓ No vulnerable initialization functions found

✓ No risky data handling found

✓ No number accuracy bug found

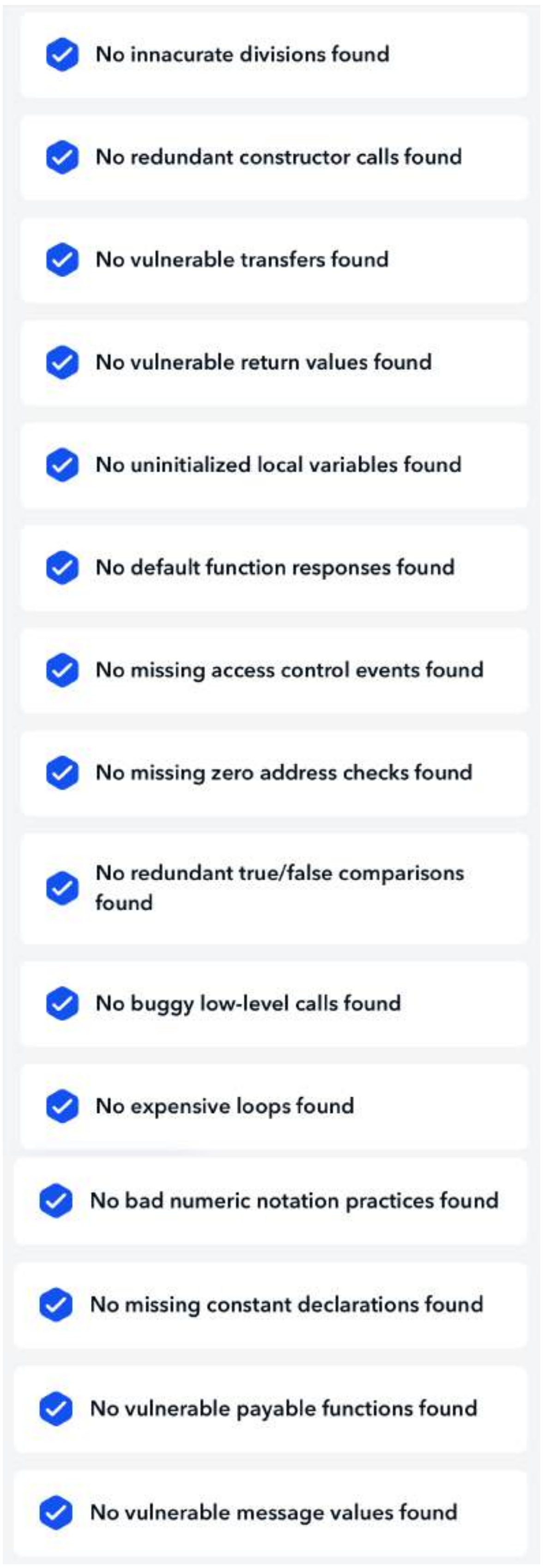
✓ No out-of-range number vulnerability found

✓ No map data deletion vulnerabilities found

✓ No tautologies or contradictions found

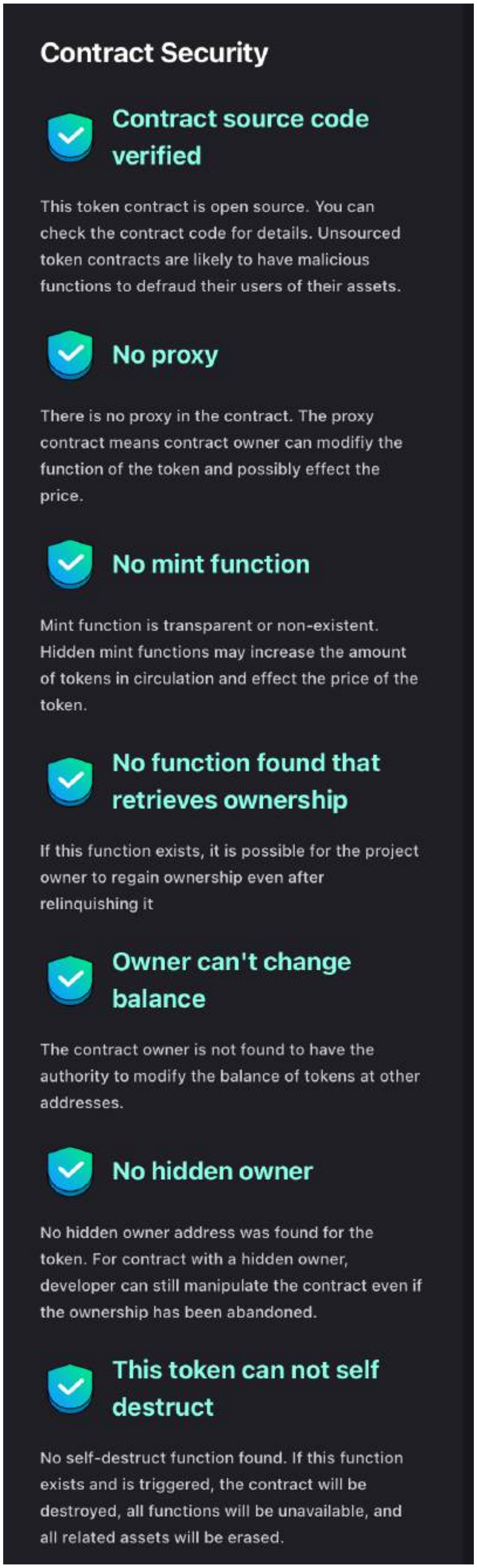
✓ No faulty true/false values found

```
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `from` to `to` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) external returns (bool);
}

// File @openzeppelin/contracts/token/ERC20/extensions/
IERC20Metadata.sol@v4.8.1
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/
IERC20Metadata.sol)

/**
 * @dev Interface for the optional metadata functions from the ERC20
standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
```



- ✓ No innacurate divisions found
- ✓ No redundant constructor calls found
- ✓ No vulnerable transfers found
- ✓ No vulnerable return values found
- ✓ No uninitialized local variables found
- ✓ No default function responses found
- ✓ No missing access control events found
- ✓ No missing zero address checks found
- ✓ No redundant true/false comparisons found
- ✓ No buggy low-level calls found
- ✓ No expensive loops found
- ✓ No bad numeric notation practices found
- ✓ No missing constant declarations found
- ✓ No vulnerable payable functions found
- ✓ No vulnerable message values found

```
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}

// File @openzeppelin/contracts/token/ERC20/ERC20.sol@v4.8.1

// OpenZeppelin Contracts (last updated v4.8.0) (token/ERC20/ERC20.sol)

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using
{_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.openzeppelin.com/t/how-to-implement-erc20-supply-
mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions
revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of ERC20
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
```



Contract Security

Contract source code verified

This token contract is open source. You can check the contract code for details. Unsourced token contracts are likely to have malicious functions to defraud their users of their assets.

No proxy

There is no proxy in the contract. The proxy contract means contract owner can modifiy the function of the token and possibly effect the price.

No mint function

Mint function is transparent or non-existent. Hidden mint functions may increase the amount of tokens in circulation and effect the price of the token.

No function found that retrieves ownership

If this function exists, it is possible for the project owner to regain ownership even after relinquishing it

Owner can't change balance

The contract owner is not found to have the authority to modify the balance of tokens at other addresses.

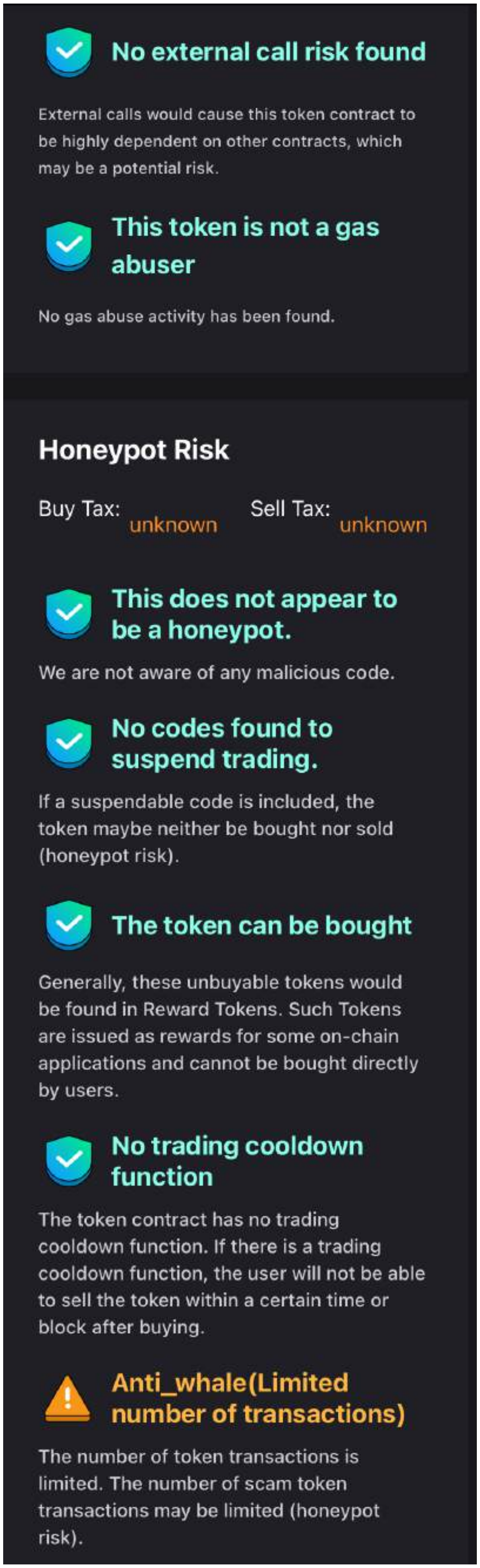
No hidden owner

No hidden owner address was found for the token. For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned.

This token can not self destruct

No self-destruct function found. If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.

```
/**
 * @dev Sets the values for {name} and {symbol}.
 *
 * The default value of {decimals} is 18. To select a different value for
 * {decimals} you should overload it.
 *
 * All two of these values are immutable: they can only be set once during
 * construction.
 */
constructor(string memory name_, string memory symbol_) {
    _name = name_;
    _symbol = symbol_;
}

/**
 * @dev Returns the name of the token.
 */
function name() public view virtual override returns (string memory) {
    return _name;
}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view virtual override returns (string memory) {
    return _symbol;
}

/**
  * @dev Returns the number of decimals used to get its user
representation.
  * For example, if `decimals` equals `2`, a balance of `505` tokens should
  * be displayed to a user as `5.05` (`505 / 10 ** 2`).
  *
  * Tokens usually opt for a value of 18, imitating the relationship between
  * Ether and Wei. This is the value {ERC20} uses, unless this function is
  * overridden;
  *
  * NOTE: This information is only used for _display_ purposes: it in
  * no way affects any of the arithmetic of the contract, including
  * {IERC20-balanceOf} and {IERC20-transfer}.
  */
function decimals() public view virtual override returns (uint8) {
    return 18;
}

/**
 * @dev See {IERC20-totalSupply}.
 */
```


No external call risk found

External calls would cause this token contract to be highly dependent on other contracts, which may be a potential risk.

This token is not a gas abuser

No gas abuse activity has been found.

Honeypot Risk

Buy Tax: unknown Sell Tax: unknown

This does not appear to be a honeypot.

We are not aware of any malicious code.

No codes found to suspend trading.

If a suspendable code is included, the token maybe neither be bought nor sold (honeypot risk).

The token can be bought

Generally, these unbuyable tokens would be found in Reward Tokens. Such Tokens are issued as rewards for some on-chain applications and cannot be bought directly by users.

No trading cooldown function

The token contract has no trading cooldown function. If there is a trading cooldown function, the user will not be able to sell the token within a certain time or block after buying.

Anti_whale(Limited number of transactions)

The number of token transactions is limited. The number of scam token transactions may be limited (honeypot risk).

```
function totalSupply() public view virtual override returns (uint256) {
    return _totalSupply;
}

/**
 * @dev See {IERC20-balanceOf}.
 */
function balanceOf(
    address account
) public view virtual override returns (uint256) {
    return _balances[account];
}

/**
 * @dev See {IERC20-transfer}.
 *
 * Requirements:
 *
 * - `to` cannot be the zero address.
 * - the caller must have a balance of at least `amount`.
 */
function transfer(
    address to,
    uint256 amount
) public virtual override returns (bool) {
    address owner = _msgSender();
    _transfer(owner, to, amount);
    return true;
}

/**
 * @dev See {IERC20-allowance}.
 */
function allowance(
    address owner,
    address spender
) public view virtual override returns (uint256) {
    return _allowances[owner][spender];
}

/**
 * @dev See {IERC20-approve}.
 *
 * NOTE: If `amount` is the maximum `uint256`, the allowance is not
updated on
 * `transferFrom`. This is semantically equivalent to an infinite approval.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function approve(
    address spender,
    uint256 amount
) public virtual override returns (bool) {
    address owner = _msgSender();
    _approve(owner, spender, amount);
    return true;
}

/**
 * @dev See {IERC20-transferFrom}.
 *
 * Emits an {Approval} event indicating the updated allowance. This is not
 * required by the EIP. See the note at the beginning of {ERC20}.
 *
 * NOTE: Does not update the allowance if the current allowance
 * is the maximum `uint256`.
 *
 * Requirements:
 *
 * - `from` and `to` cannot be the zero address.
 * - `from` must have a balance of at least `amount`.
 * - the caller must have allowance for ``from``'s tokens of at least
 * `amount`.
 */
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);
    return true;
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {IERC20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function increaseAllowance(
    address spender,
    uint256 addedValue
) public virtual returns (bool) {
    address owner = _msgSender();
    _approve(owner, spender, allowance(owner, spender) + addedValue);
    return true;
}

/**
 * @dev Atomically decreases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {IERC20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 * - `spender` must have allowance for the caller of at least
 * `subtractedValue`.
 */
function decreaseAllowance(
    address spender,
    uint256 subtractedValue
) public virtual returns (bool) {
    address owner = _msgSender();
    uint256 currentAllowance = allowance(owner, spender);
    require(
        currentAllowance >= subtractedValue,
        "ERC20: decreased allowance below zero"
    );
    unchecked {
        _approve(owner, spender, currentAllowance - subtractedValue);
    }

    return true;
}

/**
 * @dev Moves `amount` of tokens from `from` to `to`.
 *
 * This internal function is equivalent to {transfer}, and can be used to
 * e.g. implement automatic token fees, slashing mechanisms, etc.
 *
 * Emits a {Transfer} event.
 *
 * Requirements:
 *
 * - `from` cannot be the zero address.
 * - `to` cannot be the zero address.
 * - `from` must have a balance of at least `amount`.
 */
function _transfer(
    address from,
    address to,
    uint256 amount
) internal virtual {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(from, to, amount);

    uint256 fromBalance = _balances[from];
    require(
        fromBalance >= amount,
        "ERC20: transfer amount exceeds balance"
    );
    unchecked {
        _balances[from] = fromBalance - amount;
        // Overflow not possible: the sum of all balances is capped by totalSupply, and the sum is preserved by
        // decrementing then incrementing.
        _balances[to] += amount;
    }

    emit Transfer(from, to, amount);

    _afterTokenTransfer(from, to, amount);
}

/** @dev Creates `amount` tokens and assigns them to `account`, increasing
 * the total supply.
 *
 * Emits a {Transfer} event with `from` set to the zero address.
 *
 * Requirements:
 *
 * - `account` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    unchecked {
        // Overflow not possible: balance + amount is at most totalSupply + amount, which is checked above.
        _balances[account] += amount;
    }
    emit Transfer(address(0), account, amount);
    _afterTokenTransfer(address(0), account, amount);
}

/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements:
 *
 * - `account` cannot be the zero address.
 * - `account` must have at least `amount` tokens.
 */
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];
    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);

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

/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
 *
 * This internal function is equivalent to `approve`, and can be used to
 * e.g. set automatic allowances for certain subsystems, etc.
 *
 * Emits an {Approval} event.
 *
 * Requirements:
 *
 * - `owner` cannot be the zero address.
 * - `spender` cannot be the zero address.
 */
function _approve(
    address owner,
    address spender,
    uint256 amount
```

```
) internal virtual {
    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);
}

/**
 * @dev Updates `owner` s allowance for `spender` based on spent `amount`.
 *
 * Does not update the allowance amount in case of infinite allowance.
 * Revert if not enough allowance is available.
 *
 * Might emit an {Approval} event.
 */
function _spendAllowance(
    address owner,
    address spender,
    uint256 amount
) internal virtual {
    uint256 currentAllowance = allowance(owner, spender);
    if (currentAllowance != type(uint256).max) {
        require(
            currentAllowance >= amount,
            "ERC20: insufficient allowance"
        );
        unchecked {
            _approve(owner, spender, currentAllowance - amount);
        }
    }
}

/**
 * @dev Hook that is called before any transfer of tokens. This includes
 * minting and burning.
 *
 * Calling conditions:
 *
 * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
 * will be transferred to `to`.
 * - when `from` is zero, `amount` tokens will be minted for `to`.
 * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
 * - `from` and `to` are never both zero.
 *
 * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
 */
function _beforeTokenTransfer(
    address from,
    address to,
        uint256 amount
    ) internal virtual {}

    /**
     * @dev Hook that is called after any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * has been transferred to `to`.
     * - when `from` is zero, `amount` tokens have been minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 */

library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
```

```
}

/**
 * @dev Returns the integer division of two unsigned integers. Reverts on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator. Note: this function uses a
 * `revert` opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b) internal pure returns (uint256) {
    return div(a, b, "SafeMath: division by zero");
}

/**
 * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator. Note: this function uses a
 * `revert` opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b > 0, errorMessage);
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold

    return c;
}

/**
 * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
 * Reverts when dividing by zero.
 *
 * Counterpart to Solidity's `%` operator. This function uses a `revert`
 * opcode (which leaves remaining gas untouched) while Solidity uses an
 * invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 *
 * - The divisor cannot be zero.
 */
```

```
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

interface IUniswapV2Factory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint
    );

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function getPair(
        address tokenA,
        address tokenB
    ) external view returns (address pair);

    function allPairs(uint) external view returns (address pair);

    function allPairsLength() external view returns (uint);

    function createPair(
        address tokenA,
        address tokenB
    ) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
}

interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);

    function decimals() external pure returns (uint8);

    function totalSupply() external view returns (uint);

    function balanceOf(address owner) external view returns (uint);

    function allowance(
        address owner,
        address spender
    ) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);

    function transfer(address to, uint value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint value
    ) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);

    function PERMIT_TYPEHASH() external pure returns (bytes32);

    function nonces(address owner) external view returns (uint);

    function permit(
        address owner,
        address spender,
        uint value,
        uint deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(
        address indexed sender,
        uint amount0,
        uint amount1,
        address indexed to
    );
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);

    function factory() external view returns (address);

    function token0() external view returns (address);

    function token1() external view returns (address);

    function getReserves()
        external
        view
        returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);

    function price0CumulativeLast() external view returns (uint);

    function price1CumulativeLast() external view returns (uint);

    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);

    function burn(address to) external returns (uint amount0, uint amount1);

    function swap(
        uint amount0Out,
        uint amount1Out,
        address to,
        bytes calldata data
    ) external;

    function skim(address to) external;

    function sync() external;

    function initialize(address, address) external;
```

```
}

interface IUniswapV2Router01 {
    function factory() external pure returns (address);

    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);

    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    )
        external
        payable
        returns (uint amountToken, uint amountETH, uint liquidity);

    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);

    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);

    function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax,
    uint8 v,
    bytes32 r,
    bytes32 s
) external returns (uint amountA, uint amountB);

function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax,
    uint8 v,
    bytes32 r,
    bytes32 s
) external returns (uint amountToken, uint amountETH);

function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);

function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);

function swapExactETHForTokens(
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external payable returns (uint[] memory amounts);

function swapTokensForExactETH(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);

    function swapExactTokensForETH(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);

    function swapETHForExactTokens(
        uint amountOut,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable returns (uint[] memory amounts);

    function quote(
        uint amountA,
        uint reserveA,
        uint reserveB
    ) external pure returns (uint amountB);

    function getAmountOut(
        uint amountIn,
        uint reserveIn,
        uint reserveOut
    ) external pure returns (uint amountOut);

    function getAmountIn(
        uint amountOut,
        uint reserveIn,
        uint reserveOut
    ) external pure returns (uint amountIn);

    function getAmountsOut(
        uint amountIn,
        address[] calldata path
    ) external view returns (uint[] memory amounts);

    function getAmountsIn(
        uint amountOut,
        address[] calldata path
    ) external view returns (uint[] memory amounts);
}
```

```
interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountETH);

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
}

contract MixedCryptoArts is ERC20, Ownable {
    using SafeMath for uint256;
```

```
 uint256 public _transferTax = 4;
 address public taxWallet = 0x426f0be4102a6A1752290a2A0f965d619D792e0a;
 mapping (address => bool) public _isWhiteListedFromFee;


uint256 public maxWalletLimit = 4444444444444 * 10 ** decimals();

 IUniswapV2Router02 public immutable uniswapV2Router;
 address public immutable uniswapV2Pair;

 constructor() ERC20("Mixed Crypto Arts", "MCA") {
     IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
       0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
    );
    // CREATE A UNISWAP PAIR FOR THIS NEW TOKEN
    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());
    // SET THE REST OF THE CONTRACT VARIABLES
    uniswapV2Router = _uniswapV2Router;

    // MINT INITIAL SUPPLY
    _mint(msg.sender, 888888888888888 * 10 ** decimals());

    //Exclude owner from fees
    _isWhiteListedFromFee[owner()] = true;
    _isWhiteListedFromFee[taxWallet] = true;
 }

 function _transfer(
    address sender,
    address recipient,
    uint256 amount
 ) internal virtual override {
    uint256 transferAmount = amount;

    if(!_isWhiteListedFromFee[sender] && !_isWhiteListedFromFee[recipient]){

    if(recipient != uniswapV2Pair){
        require(balanceOf(recipient) + amount <= maxWalletLimit, "Exceeds the maxWalletLimit.");
    }

    if(_transferTax > 0) {
         uint256 fee  = amount.mul(_transferTax).div(100);
         transferAmount=amount.sub(fee);
          super._transfer(sender, taxWallet, fee);
  }
    }
    super._transfer(sender, recipient, transferAmount);
 }
```

```
  function changeTaxWallet(address newTaxWallet) public onlyOwner {
   require(newTaxWallet != address(0), "New tax wallet cannot be zero address");
   require(newTaxWallet != taxWallet, "New tax wallet must be different from current");
   _isWhiteListedFromFee[taxWallet] = false;
   taxWallet = newTaxWallet;
   _isWhiteListedFromFee[newTaxWallet] = true;
  }

   function whiteListFromFee(address account) public onlyOwner {
      _isWhiteListedFromFee[account] = true;
   }

   function includeInFee(address account) public onlyOwner{
      _isWhiteListedFromFee[account] = false;
   }

   function changeMaxWalletLimit(uint256 _limit) public onlyOwner{
      require(_limit > totalSupply().div(200),"Limit too low");
      maxWalletLimit = _limit;
   }

   function changeTaxes(uint256 transferTax) public onlyOwner {
        require(transferTax <= 25, "Tax too high");
      _transferTax = transferTax;

   }
   function burnTokens(uint256 amount) public onlyOwner {
      _burn(msg.sender, amount);
   }
}}
```

Blockchain Council

CERTIFIED SMART CONTRACT AUDITOR™

THIS IS TO ACKNOWLEDGE THAT

HAS SUCCESSFULLY COMPLETED ALL REQUIREMENTS & CRITERIA FOR

CERTIFIED SMART CONTRACT AUDITOR™

CERTIFICATION THROUGH EXAMINATION & TRAINING

ADMINISTERED BY BLOCKCHAIN COUNCIL

Certified

CERTIFICATE ID: XXXXXXX
ISSUED ON: FEB 28, 2025

TOSHENDRA SHARMA
EXECUTIVE DIRECTOR, BLOCKCHAIN COUNCIL

CERTIFIED
CRYPTO
CURRENCY
AUDITOR™
Certified
Certified
Blockchain Expert™
MIT MANAGEMENT SLOAN SCHOOL
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
SLOAN SCHOOL OF MANAGEMENT

Terrence Nibbles, CCE, CCA
Auditor #17865