American Cannabis Society (ACST) Token

Smart Contract Audit

ACST

Terrance Nibbles - Certified Auditor

December 5, 2023

(ACST) Token

Smart Contract TESTnet Audit

Preface

This audit is of the ACST TESTnet contract that was provided for detailed analysis in November 2023. The entire solidity smart contract code is listed at the end of the report. The security issues described within the report can be corrected prior to deployment on the MAINnet.

Website: http://ACST.IO
Legacy Token: https://reefertoken.io
Facebook: https://www.facebook.com/profile.php?id=100077209938066

DISCLAIMER:

Disclaimer:

This audit report is based on a professional review of the provided smart contract deployed on the TESTnet. 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.

The provided Solidity code appears to be a basic ERC-20 token implementation. I'll go through the code and provide a detailed review, highlighting potential vulnerabilities and explaining the functions.

PROPOSED MAX TOTAL SUPPLY

100,000,000,000
ACST
POLYGON Blockchain

Review of TESTNET (ACST) Token Smart Contract

Smart Contract Review and Function Breakdown

Summary:

The provided smart contract is an ERC-20 token named "NewToken." It inherits from the OpenZeppelin contracts IERC20 and Ownable. Below is a breakdown of the key functions and a review of potential vulnerabilities.

Key Functions:

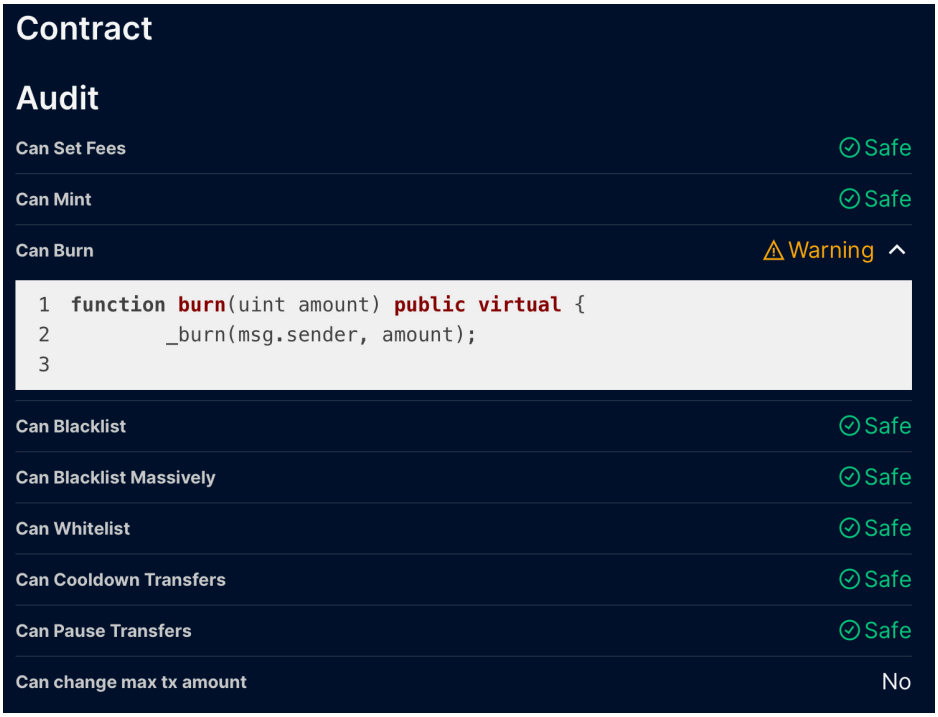
- Constructor:
  - Initializes the total supply of the token and allocates the entire supply to the contract deployer (owner).
- ERC-20 Standard Functions:
  - Implements standard ERC-20 functions such as balanceOf, transfer, allowance, approve, transferFrom, increaseAllowance, and decreaseAllowance.
  - The transfer, transferFrom, and approve functions include necessary checks and emit events.

- _approve Function:
  - Internal function to update allowance and emit the Approval event.
- mint Function:
  - Allows the owner to mint new tokens and allocate them to a specified address.
  - Ensures the minted tokens are not sent to the zero address.
- setTaxRates Function:
  - Allows the owner to set buy and sell tax rates.
  - Ensures that the total tax rates (buyTax + sellTax) do not exceed 100%.
- convertReeferHolders Function:
  - Allows the owner to trigger a function to convert Reefer token holders to NewToken.
  - The implementation of this function is left for the developer to define.

Potential Vulnerabilities and Recommendations:

- Visibility of State Variables:
  - The state variables balances and allowances are currently private. Ensure that their visibility is appropriate for your use case.
- Reentrancy:
  - The contract does not explicitly use a reentrancy guard. Consider adding a modifier or using the ReentrancyGuard from OpenZeppelin to prevent reentrancy attacks.
- Total Supply Overflow:
  - Be cautious with total supply modifications. Ensure that additions to totalSupply cannot cause overflow.
-
-

- Gas Limit:
    - Ensure that gas limits are sufficient, especially for functions like mint that modify state variables and execute external operations.
- Default Visibility:
    - Explicitly declare the visibility of functions and state variables. Although Solidity 0.8.x applies certain defaults, it's good practice to be explicit.
- Event Data:
    - Ensure that event data provides sufficient information for users and dApps to track token movements.
-
- Unused Functions:
    - If certain functions are not intended for use, consider removing or commenting on their purpose.
- Documentation:
    - Add comments and documentation to clarify the purpose and usage of each function, especially for custom functions like convertReeferHolders.



- No vulnerable withdrawal functions found
- No reentrancy risk found
- Contract owner cannot abuse ERC20 approvals
- 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.
- Contract was not deployed recently.
- No mixers utilized by contract deployer.

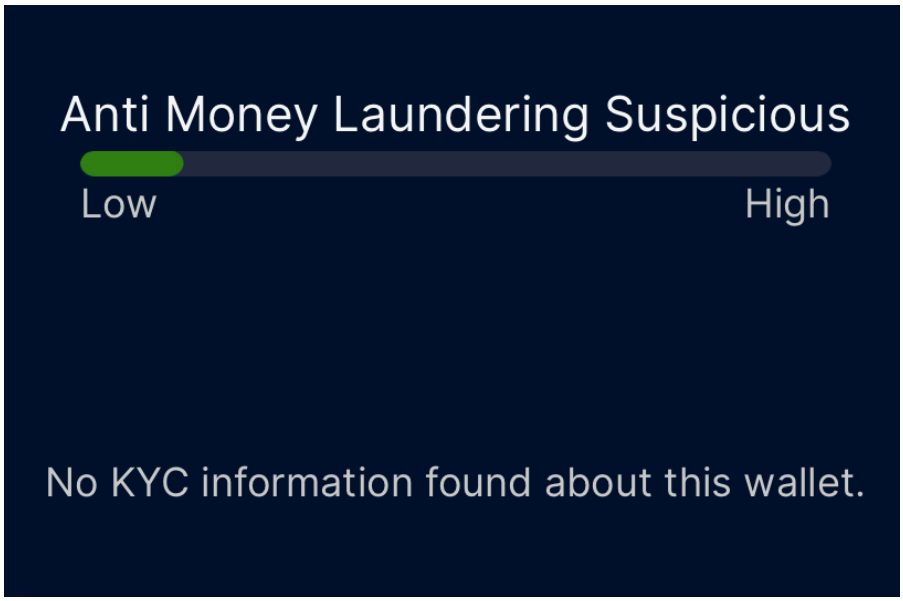
SMART CONTRACT CODE.SOL PROVIDED FOR REVIEW: ACST (token)

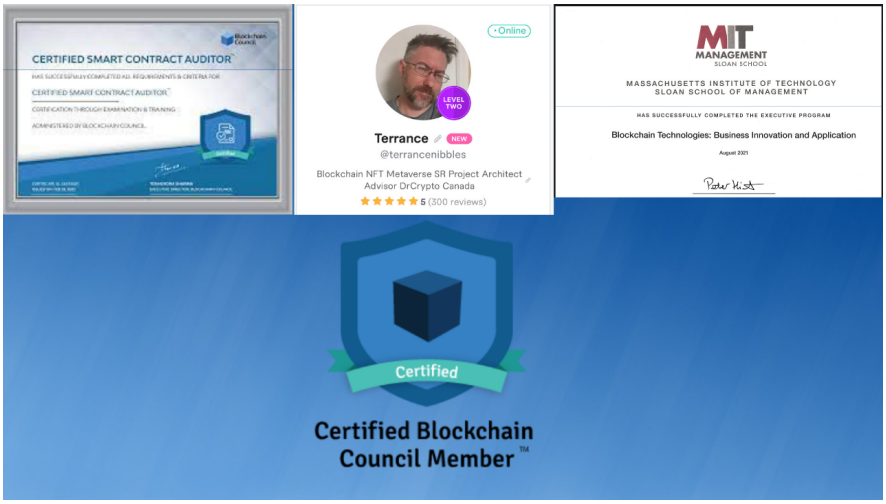
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract NewToken is IERC20, Ownable {
    string public name = "NewToken";
    string public symbol = "NT";
    uint8 public decimals = 18;
    uint256 public totalSupply = 100000000 * 10 ** uint256(decimals);

    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private allowances;

    constructor() {
        balances[msg.sender] = totalSupply;
    }

    function balanceOf(address account) external view override returns (uint256) {
        return balances[account];
    }

    function transfer(address to, uint256 amount) external override returns (bool) {
        require(to != address(0), "ERC20: transfer to the zero address");
        require(balances[msg.sender] >= amount, "ERC20: transfer amount exceeds balance");

        balances[msg.sender] -= amount;
        balances[to] += amount;

        emit Transfer(msg.sender, to, amount);
        return true;
    }

    function allowance(address owner, address spender) external view override returns (uint256) {
        return allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) external override returns (bool) {
        allowances[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);
        return true;
    }
```

- No instances of native token drainage upon revoking tokens were detected in the contract.
- Securely hardcoded Uniswap router ensuring protection against router alterations.
- AI model detects robust, genuine token and user activity, earning a high score, indicating trustworthiness and community integrity.
- 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 significant creator rugpull risk found.

```
function transferFrom(address from, address to, uint256 amount) external override returns (bool) {
    require(to != address(0), "ERC20: transfer to the zero address");
    require(balances[from] >= amount, "ERC20: transfer amount exceeds balance");
    require(allowances[from][msg.sender] >= amount, "ERC20: allowance too low");

    balances[from] -= amount;
    balances[to] += amount;
    allowances[from][msg.sender] -= amount;

    emit Transfer(from, to, amount);
    return true;
}

function increaseAllowance(address spender, uint256 addedValue) external returns (bool) {
    uint256 newAllowance = allowances[msg.sender][spender] + addedValue;
    _approve(msg.sender, spender, newAllowance);
    return true;
}

function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool) {
    uint256 currentAllowance = allowances[msg.sender][spender];
    require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
    uint256 newAllowance = currentAllowance - subtractedValue;
    _approve(msg.sender, spender, newAllowance);
    return true;
}

function _approve(address owner, address spender, uint256 amount) internal {
    allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}

function mint(address to, uint256 amount) external onlyOwner {
    require(to != address(0), "ERC20: mint to the zero address");
    totalSupply += amount;
    balances[to] += amount;
    emit Transfer(address(0), to, amount);
}

function setTaxRates(uint8 buyTax, uint8 sellTax) external onlyOwner {
    require(buyTax + sellTax <= 100, "Total tax exceeds 100%");


        // Your logic to set the tax rates and distribute to wallets here
        }

        function convertReeferHolders() external onlyOwner {
        // Your logic to convert Reefer token holders to NewToken here
        }
```

- **No adjustable maximum supply found.**
- **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.**





Terrence Nibbles, CCE, CCA Auditor #17865