ecoSave

ACT. SAVE. GROW.

Ecosave Token Pre-Sale

Table of Contents

1. Introduction

The EcoSavePresale smart contract facilitates the sale of EcoSave tokens in exchange for USDT (Tether). The contract is designed with security and transparency in mind, utilizing OpenZeppelin libraries for reentrancy protection, safe token transfers, and ownership control. The presale contract ensures that users can purchase EcoSave tokens securely and reliably by enforcing critical safety checks and maintaining precision during token purchases.

2. Contract Overview

The EcoSavePresale contract implements ERC20 token purchases using USDT with a fixed price of 0.10 USDT per token. The contract ensures security and owner control over key functions such as withdrawing funds from the presale.

3. Key Features and Functionalities

I. Token Purchase

The main function of the contract is to allow users to buy EcoSave tokens using USDT. The amount of tokens a user can purchase is calculated based on the amount of USDT sent, using a predefined price per token.

II. Security Mechanisms

The contract includes several security mechanisms to ensure safe operation, including:
- Reentrancy Protection: The ReentrancyGuard modifier is used to prevent reentrancy attacks.
- SafeERC20: The SafeERC20 library is used for secure token transfers, ensuring no loss of funds due to errors.
- Ownership Control: The contract is Ownable, ensuring only the owner can perform sensitive operations such as withdrawing tokens and USDT.

4. Why It Is Safe to Use the Contract in the Presale

The EcoSavePresale contract follows best practices in Solidity development and uses OpenZeppelin libraries to enhance security. Features such as the ReentrancyGuard modifier, SafeERC20 library, and owner-only function access ensure that the contract is secure from common vulnerabilities and safe for presale operations. The immutability of smart contracts ensures that the code cannot be altered once deployed, making the presale transparent and reliable for participants.

5. Conclusion

The EcoSavePresale smart contract offers a secure and reliable platform for the sale of EcoSave tokens. By employing industry-standard security mechanisms and adhering to best practices, the contract ensures that presale participants can safely purchase tokens. The combination of reentrancy protection, secure token transfers, and owner control makes this contract an ideal choice for managing token sales during the presale phase.

Smart Contract Address:

0xcfcd8d07cc390889845b2f73087594cc6de73867

6. Code

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

// Import OpenZeppelin contracts with specified versions
import "@openzeppelin/contracts@4.9.0/access/Ownable2Step.sol";
```

```
import "@openzeppelin/contracts@4.9.0/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts@4.9.0/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts@4.9.0/utils/math/Math.sol";

contract EcoSavePresale is Ownable2Step, ReentrancyGuard {
  using SafeERC20 for IERC20;

  IERC20 public immutable ecosaveToken;
  IERC20 public immutable usdtToken;
  uint256 public immutable pricePerToken; // Price per EcoSave token in USDT (6
decimals)

  event TokensPurchased(address indexed buyer, uint256 usdtAmount, uint256
ecosaveAmount);
  event USDTWithdrawn(uint256 amount);
  event EcoSaveWithdrawn(uint256 amount);

  constructor(address _ecosaveToken, address _usdtToken)
    payable
  {
    require(_ecosaveToken != address(0), "Invalid EcoSave token address");
    require(_usdtToken != address(0), "Invalid USDT token address");

    ecosaveToken = IERC20(_ecosaveToken);
    usdtToken = IERC20(_usdtToken);
    pricePerToken = 100_000; // 0.10 USDT per EcoSave token (USDT has 6 decimals)
  }

  modifier nonZeroAmount(uint256 amount) {
    require(amount != 0, "Amount must be greater than zero");
    _;
  }

  function buyTokens(uint256 usdtAmount)
    external
    nonReentrant
    nonZeroAmount(usdtAmount)
  {
    // Calculate the number of EcoSave tokens to send using mulDiv to prevent
precision loss
    uint256 ecosaveAmount = Math.mulDiv(usdtAmount, 1e18, pricePerToken);

    uint256 contractBalance = ecosaveToken.balanceOf(address(this));
    require(
      contractBalance >= ecosaveAmount,
      "Not enough EcoSave tokens left"
    );

    // Transfer USDT from buyer to the contract
```

```
        usdtToken.safeTransferFrom(msg.sender, address(this), usdtAmount);

        // Transfer EcoSave tokens to the buyer
        ecosaveToken.safeTransfer(msg.sender, ecosaveAmount);

        emit TokensPurchased(msg.sender, usdtAmount, ecosaveAmount);
    }

    function withdrawUSDT(uint256 amount)
        external
        payable
        nonReentrant
        onlyOwner
        nonZeroAmount(amount)
    {
        usdtToken.safeTransfer(owner(), amount);
        emit USDTWithdrawn(amount);
    }

    function withdrawEcoSave(uint256 amount)
        external
        payable
        nonReentrant
        onlyOwner
        nonZeroAmount(amount)
    {
        ecosaveToken.safeTransfer(owner(), amount);
        emit EcoSaveWithdrawn(amount);
    }
}
```