SWORD AVAX

PRESALE Smart Contract Audit

https://swordavax.com

Terrance Nibbles - Certified Auditor

February 27, 2024

SWORD AVAX

PRESALE Smart Contract Audit

Preface

This audit is of the SWORD AVAX contract that was provided for detailed analysis on February 26, 2024. The entire 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 Avalanche Blockchain:

https://subnets.avax.network/c-chain/address/0x9c7F474DA5cBDE24db7f32C4527c7962272347F5

And also works in conjunction with;
New Token: https://subnets.avax.network/c-chain/address/0x66B7Cd2046C633196Cfd061Db21Ca6A23Ab9ba3a

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 or Avalanche 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 / Avalanche 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.

Audit Report: Sword PRESALE Smart Contract

The SwordAvaxPresale contract is designed to manage a presale event for an ERC-20 token on the Ethereum blockchain. This contract allows whitelisted addresses to purchase tokens at a predetermined price during different phases of the presale, with each phase having specific maximum purchase limits per wallet and token prices. It incorporates features such as whitelisting through Merkle proofs, reentrancy protection, and phased purchase limits and pricing. The contract is owned and can be managed by the owner, providing administrative functions like updating sales parameters and withdrawing funds.

Features Breakdown

- Phased Presale: The contract supports four phases of sales, each with unique maximum purchase limits and token prices.
- Whitelisting: Utilizes Merkle proofs to verify whitelisted addresses for each phase, enhancing security and ensuring controlled participation.
- Reentrancy Protection: Inherits ReentrancyGuard from OpenZeppelin to prevent reentrancy attacks.
- Token Purchase: Allows whitelisted addresses to purchase tokens according to the phase-specific limits and prices, with an added bonus mechanism for larger purchases.
- Administrative Functions: Includes functions for the owner to update sale parameters, manage whitelists, and withdraw funds or tokens from the contract.

Security Considerations

- Reentrancy Guard: The use of ReentrancyGuard mitigates the risk of reentrancy attacks, a common vulnerability in contracts that handle Ether transfers.
- Whitelist Verification: Using Merkle proofs for whitelisting is an efficient and secure method to manage access without storing a large list on-chain.
- Ownership Controls: The contract inherits Ownable for managing sensitive functions, which restricts critical operations to the contract owner, minimizing unauthorized access risks.
- Validation Checks: The contract includes multiple checks for parameters like sale phase, payment amounts, and purchase limits, which helps prevent erroneous transactions and enforce sale rules.

Potential Vulnerabilities

- Smart Contract Logic: Ensure thorough testing of the logic, especially around phase transitions, token pricing, and bonus calculations, to prevent unintended behavior.
- Dependency Risks: Relies on OpenZeppelin's contracts (Ownable, ReentrancyGuard, IERC20), which are reputable but should always be kept up-to-date with the latest secure versions.
- Front-Running Risks: Public blockchain transactions are susceptible to front-running; sensitive operations (like phase changes or whitelisting updates) could be targeted. Strategies to mitigate this include using private transactions or time locks for critical operations.

Recommendations

- Comprehensive Testing: Conduct extensive testing, including unit tests and integration tests, to cover all functionalities and edge cases.
- Monitor Dependency Updates: Regularly check for updates or security advisories related to the OpenZeppelin contracts and other dependencies.

Conclusion

The SwordAvaxPresale contract implements a structured and phased approach to a token presale, incorporating essential security practices.

NO HIGH RISK ISSUES IDENTIFIED

- No vulnerable withdrawal functions found
- No reentrancy risk found
- No locks detected
- Contract cannot be upgraded
- No ERC20 approval vulnerability found
- Contract owner cannot abuse ERC20 approvals
- No blocking loops found
- Wallets cannot be blacklisted from any specific contract functionality
- No functionality can be paused by the contract owner
- No approval restrictions found
- No vulnerable ownership functions found.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.22;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/utils/cryptography/MerkleProof.sol";
import "@openzeppelin/contracts/utils/Strings.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract SwordAvaxPresale is Ownable, ReentrancyGuard {

    using Strings for uint;

    enum Step {
        Off,
        On
    }

    string public baseURI;

    Step public sellingStep;

    IERC20 public token = IERC20(0xCA419c0D70509Ac5aE8affe9D22A8Dd302B041c1);

    uint256 public MAX_SUPPLY = 3600000000;
    uint256 public totalSold;

    uint256 public MAX_PER_WALLET_PHASE_1 = 22500000;
    uint256 public MAX_PER_WALLET_PHASE_2 = 25000000;
    uint256 public MAX_PER_WALLET_PHASE_3 = 27500000;
    uint256 public MAX_PER_WALLET_PHASE_4 = 30000000;

    uint256 public PRICE_PER_TOKEN_PHASE_1 = 0.00000090 ether;
    uint256 public PRICE_PER_TOKEN_PHASE_2 = 0.00000100 ether;
    uint256 public PRICE_PER_TOKEN_PHASE_3 = 0.00000110 ether;
    uint256 public PRICE_PER_TOKEN_PHASE_4 = 0.00000120 ether;

    uint256 public MIN_PAYMENT = 0.1 ether;
```

- No retrievable ownership found.
- No mixers utilized by contract deployer.
- No previous scams by owner's wallet found.
- Recent Interaction was within 30 Days.
  Smart contract with recent user interactions, active use, and operational functionality, not abandoned.
- 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.

```
    uint256 public saleStartTime = 1709046000;

    bytes32 public merkleRootWL_PHASE_1;
    bytes32 public merkleRootWL_PHASE_2;
    bytes32 public merkleRootWL_PHASE_3;
    bytes32 public merkleRootWL_PHASE_4;

    mapping(address => uint) public amountPurchasedPhase1;
    mapping(address => uint) public amountPurchasedPhase2;
    mapping(address => uint) public amountPurchasedPhase3;
    mapping(address => uint) public amountPurchasedPhase4;

    constructor(address _initialOwner)
    Ownable(_initialOwner)
    {
        sellingStep = Step.On;
    }

  function purchase(address _account, uint _quantity, uint phase, bytes32[] calldata
_proof) external payable nonReentrant  {
        require(sellingStep == Step.On, "Sale is stopped");
        require(block.timestamp > saleStartTime, "Public sale is not activated");
        require(phase == 1 || phase == 2 || phase == 3 || phase == 4, "Incorrect phase
provided");
        require(totalSold + _quantity <= MAX_SUPPLY, "Trying to purchase more than
available");
        require(msg.value >= MIN_PAYMENT, "Minimum payment not reaced");
        if(phase == 1){
            require(isWhitelistedPhase1(_account, _proof), "Not whitelisted");
            require(amountPurchasedPhase1[_account] + _quantity <=
MAX_PER_WALLET_PHASE_1, "Max per wallet limit reached");
            require(msg.value >= PRICE_PER_TOKEN_PHASE_1 * _quantity, "Incorrect
payment");
            amountPurchasedPhase1[_account] += _quantity;
        } else if (phase == 2){
            require(isWhitelistedPhase2(_account, _proof), "Not whitelisted");
            require(amountPurchasedPhase2[_account] + _quantity <=
MAX_PER_WALLET_PHASE_2, "Max per wallet limit reached");
```

- 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
- No risky data allocation found

```
        require(msg.value >= PRICE_PER_TOKEN_PHASE_2 * _quantity,
"Incorrect payment");
        amountPurchasedPhase2[_account] += _quantity;
    } else if (phase == 3){
        require(isWhitelistedPhase3(_account, _proof), "Not whitelisted");
        require(amountPurchasedPhase3[_account] + _quantity <=
MAX_PER_WALLET_PHASE_3, "Max per wallet limit reached");
        require(msg.value >= PRICE_PER_TOKEN_PHASE_3 * _quantity,
"Incorrect payment");
        amountPurchasedPhase3[_account] += _quantity;
    } else if (phase == 4){
        require(isWhitelistedPhase4(_account, _proof), "Not whitelisted");
        require(amountPurchasedPhase4[_account] + _quantity <=
MAX_PER_WALLET_PHASE_4, "Max per wallet limit reached");
        require(msg.value >= PRICE_PER_TOKEN_PHASE_4 * _quantity,
"Incorrect payment");
        amountPurchasedPhase4[_account] += _quantity;
    }

     // Calculate the bonus based on the quantity
    uint256 bonusPercentage = 0;
    if(_quantity >= 15000000) {
        bonusPercentage = 15;
    } else if(_quantity >= 10000000) {
        bonusPercentage = 10;
    } else if(_quantity >= 5000000) {
        bonusPercentage = 5;
    }

    uint256 bonusTokens = _quantity * bonusPercentage / 100;
    uint256 totalAmountToTransfer = _quantity * 1e18 + bonusTokens *
1e18; // Assuming the token has 18 decimals

    require(token.transfer(msg.sender, totalAmountToTransfer), "Token transfer
failed");
  }
```

- 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
- No innacurate divisions found
- No redundant constructor calls found
- No vulnerable transfers found
- No vulnerable return values found

```
function setStep(uint _step) external onlyOwner {
    sellingStep = Step(_step);
}

function setSaleStartTime(uint256 _startTime) external onlyOwner {
    saleStartTime = _startTime;
}

function updateTokenAddress(IERC20 _newToken) external onlyOwner {
    token = _newToken;
}

function updateMaxSupply(uint256 _MAX_SUPPLY) external onlyOwner{
    MAX_SUPPLY = _MAX_SUPPLY;
}

function updateMaxPerWalletPhase1(uint256 _MAX_PER_WALLET_PHASE_1) external
onlyOwner{
    MAX_PER_WALLET_PHASE_1 = _MAX_PER_WALLET_PHASE_1;
}

function updateMaxPerWalletPhase2(uint256 _MAX_PER_WALLET_PHASE_2) external
onlyOwner{
    MAX_PER_WALLET_PHASE_2 = _MAX_PER_WALLET_PHASE_2;
}

function updateMaxPerWalletPhase3(uint256 _MAX_PER_WALLET_PHASE_3) external
onlyOwner{
    MAX_PER_WALLET_PHASE_3 = _MAX_PER_WALLET_PHASE_3;
}

function updateMaxPerWalletPhase4(uint256 _MAX_PER_WALLET_PHASE_4) external
onlyOwner{
    MAX_PER_WALLET_PHASE_4 = _MAX_PER_WALLET_PHASE_4;
}
```

- No vulnerable transfers found
- No vulnerable return values found
- No uninitialized local variables found
- No default function responses found
- No missing arithmetic events found
- No missing access control events found
- No missing zero address checks found
- No redundant true/false comparisons found
- No state variables vulnerable through function calls found
- No buggy low-level calls found
- No invalid solidity versions found
- No expensive loops found

```
  function updatePricePerTokenPhase1(uint256 _PRICE_PER_TOKEN_PHASE_1) external
onlyOwner{
    PRICE_PER_TOKEN_PHASE_1 = _PRICE_PER_TOKEN_PHASE_1;
  }

  function updatePricePerTokenPhase2(uint256 _PRICE_PER_TOKEN_PHASE_2) external
onlyOwner{
    PRICE_PER_TOKEN_PHASE_2 = _PRICE_PER_TOKEN_PHASE_2;
  }

  function updatePricePerTokenPhase3(uint256 _PRICE_PER_TOKEN_PHASE_3) external
onlyOwner{
    PRICE_PER_TOKEN_PHASE_3 = _PRICE_PER_TOKEN_PHASE_3;
  }

  function updatePricePerTokenPhase4(uint256 _PRICE_PER_TOKEN_PHASE_4) external
onlyOwner{
    PRICE_PER_TOKEN_PHASE_4 = _PRICE_PER_TOKEN_PHASE_4;
  }

  function updateMinPayment(uint256 _MIN_PAYMENT) external onlyOwner{
    MIN_PAYMENT = _MIN_PAYMENT;
  }

  //Whitelist
  function setMerkleRootWLPhase1(bytes32 _merkleRootWL) external onlyOwner {
    merkleRootWL_PHASE_1 = _merkleRootWL;
  }
  function setMerkleRootWLPhase2(bytes32 _merkleRootWL) external onlyOwner {
    merkleRootWL_PHASE_2 = _merkleRootWL;
  }
  function setMerkleRootWLPhase3(bytes32 _merkleRootWL) external onlyOwner {
    merkleRootWL_PHASE_3 = _merkleRootWL;
  }
  function setMerkleRootWLPhase4(bytes32 _merkleRootWL) external onlyOwner {
    merkleRootWL_PHASE_4 = _merkleRootWL;
  }
```

- No bad numeric notation practices found
- No missing constant declarations found
- No missing external function declarations found
- No vulnerable payable functions found
- No vulnerable message values found

```
    function isWhitelistedPhase1(address _account, bytes32[] calldata _proof) internal
view returns(bool) {
        return _verifyWL1(leaf(_account), _proof);
    }
    function isWhitelistedPhase2(address _account, bytes32[] calldata _proof) internal
view returns(bool) {
        return _verifyWL2(leaf(_account), _proof);
    }
    function isWhitelistedPhase3(address _account, bytes32[] calldata _proof) internal
view returns(bool) {
        return _verifyWL3(leaf(_account), _proof);
    }
    function isWhitelistedPhase4(address _account, bytes32[] calldata _proof) internal
view returns(bool) {
        return _verifyWL4(leaf(_account), _proof);
    }

    function leaf(address _account) internal pure returns(bytes32) {
        return keccak256(abi.encodePacked(_account));
    }

    function _verifyWL1(bytes32 _leaf, bytes32[] memory _proof) internal view returns(bool) {
        return MerkleProof.verify(_proof, merkleRootWL_PHASE_1, _leaf);
    }
    function _verifyWL2(bytes32 _leaf, bytes32[] memory _proof) internal view returns(bool) {
        return MerkleProof.verify(_proof, merkleRootWL_PHASE_2, _leaf);
    }
    function _verifyWL3(bytes32 _leaf, bytes32[] memory _proof) internal view returns(bool) {
        return MerkleProof.verify(_proof, merkleRootWL_PHASE_3, _leaf);
    }
    function _verifyWL4(bytes32 _leaf, bytes32[] memory _proof) internal view returns(bool) {
        return MerkleProof.verify(_proof, merkleRootWL_PHASE_4, _leaf);
    }

    //ReleaseALL
    function withdrawTokens(uint256 _amount) external onlyOwner {
        require(token.transfer(msg.sender, _amount), "Token transfer failed");
    }
   function withdrawFunds() external onlyOwner {
        uint256 balance = address(this).balance;
        require(balance > 0, "No funds to withdraw");
        payable(msg.sender).transfer(balance);
    }
}
```

```
[
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "owner",
        "type": "address"
      }
    ],
    "name": "OwnableInvalidOwner",
    "type": "error"
  },
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "account",
        "type": "address"
      }
    ],
    "name": "OwnableUnauthorizedAccount",
    "type": "error"
  },
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": true,
        "internalType": "address",
        "name": "previousOwner",
        "type": "address"
      },
      {
        "indexed": true,
        "internalType": "address",
        "name": "newOwner",
        "type": "address"
      }
    ],
    "name": "OwnershipTransferred",
    "type": "event"
  },
  {
    "inputs": [],
    "name": "MAX_PER_WALLET_PHASE_1",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
```

```
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "MAX_PER_WALLET_PHASE_2",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "MAX_PER_WALLET_PHASE_3",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "MAX_PER_WALLET_PHASE_4",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "MAX_SUPPLY",
    "outputs": [
        {
```

```
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "MIN_PAYMENT",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "PRICE_PER_TOKEN_PHASE_1",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "PRICE_PER_TOKEN_PHASE_2",
    "outputs": [
        {
            "internalType": "uint256",
            "name": "",
            "type": "uint256"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "PRICE_PER_TOKEN_PHASE_3",
```

```
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "PRICE_PER_TOKEN_PHASE_4",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "name": "amountPurchasedPhase1",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
```

```
    "name": "amountPurchasedPhase2",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "name": "amountPurchasedPhase3",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "",
        "type": "address"
      }
    ],
    "name": "amountPurchasedPhase4",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
```

```
    "inputs": [],
    "name": "baseURI",
    "outputs": [
      {
        "internalType": "string",
        "name": "",
        "type": "string"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "merkleRootWL_PHASE_1",
    "outputs": [
      {
        "internalType": "bytes32",
        "name": "",
        "type": "bytes32"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "merkleRootWL_PHASE_2",
    "outputs": [
      {
        "internalType": "bytes32",
        "name": "",
        "type": "bytes32"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "merkleRootWL_PHASE_3",
    "outputs": [
      {
        "internalType": "bytes32",
        "name": "",
        "type": "bytes32"
      }
    ],
    "stateMutability": "view",
    "type": "function"
```

```
},
{
    "inputs": [],
    "name": "merkleRootWL_PHASE_4",
    "outputs": [
        {
            "internalType": "bytes32",
            "name": "",
            "type": "bytes32"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "owner",
    "outputs": [
        {
            "internalType": "address",
            "name": "",
            "type": "address"
        }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "address",
            "name": "_account",
            "type": "address"
        },
        {
            "internalType": "uint256",
            "name": "_quantity",
            "type": "uint256"
        },
        {
            "internalType": "uint256",
            "name": "phase",
            "type": "uint256"
        },
        {
            "internalType": "bytes32[]",
            "name": "_proof",
            "type": "bytes32[]"
        }
    ],
```

```
    "name": "purchase",
    "outputs": [],
    "stateMutability": "payable",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "renounceOwnership",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "saleStartTime",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "sellingStep",
    "outputs": [
      {
        "internalType": "enum SwordAvaxPresale.Step",
        "name": "",
        "type": "uint8"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "bytes32",
        "name": "_merkleRootWL",
        "type": "bytes32"
      }
    ],
    "name": "setMerkleRootWLPhase1",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
```

```
},
{
    "inputs": [
        {
            "internalType": "bytes32",
            "name": "_merkleRootWL",
            "type": "bytes32"
        }
    ],
    "name": "setMerkleRootWLPhase2",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "bytes32",
            "name": "_merkleRootWL",
            "type": "bytes32"
        }
    ],
    "name": "setMerkleRootWLPhase3",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "bytes32",
            "name": "_merkleRootWL",
            "type": "bytes32"
        }
    ],
    "name": "setMerkleRootWLPhase4",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_startTime",
            "type": "uint256"
        }
    ],
    "name": "setSaleStartTime",
    "outputs": [],
```

```
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_step",
        "type": "uint256"
      }
    ],
    "name": "setStep",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "token",
    "outputs": [
      {
        "internalType": "contract IERC20",
        "name": "",
        "type": "address"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "totalSold",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "address",
        "name": "newOwner",
        "type": "address"
      }
    ],
```

```
    "name": "transferOwnership",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_MAX_PER_WALLET_PHASE_1",
        "type": "uint256"
      }
    ],
    "name": "updateMaxPerWalletPhase1",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_MAX_PER_WALLET_PHASE_2",
        "type": "uint256"
      }
    ],
    "name": "updateMaxPerWalletPhase2",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_MAX_PER_WALLET_PHASE_3",
        "type": "uint256"
      }
    ],
    "name": "updateMaxPerWalletPhase3",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_MAX_PER_WALLET_PHASE_4",
        "type": "uint256"
```

```
        }
    ],
    "name": "updateMaxPerWalletPhase4",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_MAX_SUPPLY",
            "type": "uint256"
        }
    ],
    "name": "updateMaxSupply",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_MIN_PAYMENT",
            "type": "uint256"
        }
    ],
    "name": "updateMinPayment",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_PRICE_PER_TOKEN_PHASE_1",
            "type": "uint256"
        }
    ],
    "name": "updatePricePerTokenPhase1",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
```

```
            "name": "_PRICE_PER_TOKEN_PHASE_2",
            "type": "uint256"
        }
    ],
    "name": "updatePricePerTokenPhase2",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_PRICE_PER_TOKEN_PHASE_3",
            "type": "uint256"
        }
    ],
    "name": "updatePricePerTokenPhase3",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_PRICE_PER_TOKEN_PHASE_4",
            "type": "uint256"
        }
    ],
    "name": "updatePricePerTokenPhase4",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "contract IERC20",
            "name": "_newToken",
            "type": "address"
        }
    ],
    "name": "updateTokenAddress",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [],
```

```
    "name": "withdrawFunds",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_amount",
        "type": "uint256"
      }
    ],
    "name": "withdrawTokens",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
```

Terrence Nibbles, CCE, CCA Auditor #17865