# Smart Contract Security Audit Report

The SlowMist Security Team received the StakeStone team's application for smart contract security audit of the StakeStone DAO on 2025.03.28. The following are the details and results of this smart contract security audit:

**Token Name :**

StakeStone DAO

**The contract address :**

https://github.com/stakestone/dao

commit: 3e44df31c6c24705db5614429f6a9cfcc5fc5c02

Audit Scope:

```
./src
├── interfaces
│   └── IVeSTO.sol
├── token
│   ├── STO.sol
│   └── VeSTO.sol
└── utils
    └── Escrow.sol
```

**The audit items and results :**

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

| NO. | Audit Items | Result |
|---|---|---|
| 1 | Replay Vulnerability | Passed |
| 2 | Denial of Service Vulnerability | Passed |
| 3 | Race Conditions Vulnerability | Passed |
| 4 | Authority Control Vulnerability Audit | Some Risks |
| 5 | Integer Overflow and Underflow Vulnerability | Passed |
| 6 | Gas Optimization Audit | Passed |
| 7 | Design Logic Audit | Passed |
| 8 | Uninitialized Storage Pointers Vulnerability | Passed |

| NO. | Audit Items | Result |
|---|---|---|
| 9 | Arithmetic Accuracy Deviation Vulnerability | Passed |
| 10 | "False top-up" Vulnerability | Passed |
| 11 | Malicious Event Log Audit | Passed |
| 12 | Scoping and Declarations Audit | Passed |
| 13 | Safety Design Audit | Passed |
| 14 | Non-privacy/Non-dark Coin Audit | Passed |

**Audit Result :** Medium Risk

**Audit Number :** 0X002504010001

**Audit Date :** 2025.03.28 - 2025.04.01

**Audit Team :** SlowMist Security Team

**Summary conclusion :** This is the DAO contract for the StakeStone protocol, with an audit covering STO tokens, veSTO tokens, and Escrow contract. The total supply of STO tokens is fixed, while veSTO tokens have a transferRestricted function. When in transferRestricted state, token transfers are only allowed from addresses with allowedFrom status set to true, or to addresses with allowedTo status set to true. Regarding veSTO tokens, the following excessive privilege risks should be noted:

1. The MINTER_ROLE can mint unlimited tokens through the mint function.
2. The BURNER_ROLE can burn any user's tokens through the burn function.

## The source code:

STO.sol

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

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

/// @title STO: StakeStone Token for Omnichain Liquidity
contract STO is ERC20 {
    constructor(
        address _receiver
    ) ERC20("StakeStone Token for Omnichain Liquidity", "STO") {
        _mint(_receiver, 1_000_000_000 * 1e18);
    }
}
```

VeSTO.sol

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

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {AccessControl} from "@openzeppelin/contracts/access/AccessControl.sol";

/// @title veSTO: Vote-Escrowed STO Token
contract VeSTO is ERC20, AccessControl {
    /*//////////////////////////////////////////////////////////////////////////
                                STATE VARIABLES
    //////////////////////////////////////////////////////////////////////////*/

    /// @notice A role designated for minting veSTO.
    /// @dev Hash digests for `MINTER_ROLE`
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

    /// @notice A role designated for burning veSTO.
    /// @dev Hash digests for `BURNER_ROLE`
    bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");

    /// @notice If veSTO can be transferred.
    bool public transferRestricted;

    /// @notice Mapping of addresses that are allowed to transfer tokens to any
address
    mapping(address => bool) public allowedFrom;
    /// @notice Mapping of addresses that are allowed to receive tokens from any
address
    mapping(address => bool) public allowedTo;

    /*//////////////////////////////////////////////////////////////////////////
                                EVENTS
    //////////////////////////////////////////////////////////////////////////*/

    event SetAllowedFrom(address indexed from, bool isAllowedFrom);
    event SetAllowedTo(address indexed to, bool isAllowedTo);
    event TransferRestrictionsEnabled();
    event TransferRestrictionsDisabled();
    /*//////////////////////////////////////////////////////////////////////////
                                CONSTRUCTOR
    //////////////////////////////////////////////////////////////////////////*/

    constructor(address _admin) ERC20("Vote-Escrowed STO", "veSTO") {
        _grantRole(DEFAULT_ADMIN_ROLE, _admin);

        transferRestricted = true;
        emit TransferRestrictionsEnabled();
    }

    /*//////////////////////////////////////////////////////////////////////////
                                ADMIN FUNCTIONS
    //////////////////////////////////////////////////////////////////////////*/

    /// @notice This function allows the admin to set the allowedFrom status of an
address
    /// @param _from The address whose allowedFrom status is being set
    /// @param _isAllowedFrom The new allowedFrom status
    function setAllowedFrom(
        address _from,
        bool _isAllowedFrom
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        allowedFrom[_from] = _isAllowedFrom;
        emit SetAllowedFrom(_from, _isAllowedFrom);
    }

    /// @notice This function allows the admin to set the allowedTo status of an
address
    /// @param _to The address whose allowedFrom status is being set
    /// @param _isAllowedTo The new allowedFrom status
    function setAllowedTo(
        address _to,
        bool _isAllowedTo
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        allowedTo[_to] = _isAllowedTo;
        emit SetAllowedTo(_to, _isAllowedTo);
    }

    /// @notice This function allows the admin to disable transfer restrictions
    function disableTransferRestrictions()
        external
        onlyRole(DEFAULT_ADMIN_ROLE)
    {
        require(
            transferRestricted,
            "Transfer restrictions are already disabled"
        );
        transferRestricted = false;
        emit TransferRestrictionsDisabled();
    }

    /// @notice This function allows the admin to enable transfer restrictions
    function enableTransferRestrictions()
        external
        onlyRole(DEFAULT_ADMIN_ROLE)
    {
        require(
            !transferRestricted,
            "Transfer restrictions are already enabled"
        );

        transferRestricted = true;
        emit TransferRestrictionsEnabled();
    }

    /// @notice This function allows the MINTER_ROLE to mint veSTO to an address
    /// @param _to The receiver address
    /// @param _amount The amount of the token will be minted
    function mint(address _to, uint256 _amount) external onlyRole(MINTER_ROLE) {
//SlowMist// The MINTER_ROLE can arbitrarily mint veSTO tokens through the mint
function.
        _mint(_to, _amount);
    }

    /// @notice This function allows the BURNER_ROLE to burn veSTO from an address
    /// @param _from The address which veSTO will be burned from
    /// @param _amount The amount of the token will be burned
    function burn(
        address _from,
        uint256 _amount
    ) external onlyRole(BURNER_ROLE) { //SlowMist// The BURNER_ROLE can burn the
veSTO tokens of any user through the burn function.
        _burn(_from, _amount);
    }

    /*//////////////////////////////////////////////////////////////////////////
                                INTERNAL FUNCTIONS
    //////////////////////////////////////////////////////////////////////////*/

    /**
     * @notice Overrides the _update function to enforce transfer restrictions
     * @param _from the address tokens are being transferred from
     * @param _to the address tokens are being transferred to
     * @param _amount the amount of tokens being transferred
     */
    function _update(
        address _from,
        address _to,
        uint256 _amount
    ) internal override {
        if (transferRestricted) {
            require(
                _from == address(0) ||
                    _to == address(0) ||
                    allowedFrom[_from] || //SlowMist// Addresses with the allowedFrom
status set to true can transfer tokens to any user.
                    allowedTo[_to], //SlowMist// Any user can transfer tokens to
addresses with the allowedTo status set to true.
                "not in whitelist"
            );
        }
        super._update(_from, _to, _amount);
    }
}
```

Escrow.sol

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

import {Ownable2Step, Ownable} from
"@openzeppelin/contracts/access/Ownable2Step.sol";
import {TransferHelper} from "@uniswap/v3-
periphery/contracts/libraries/TransferHelper.sol";

import {IVeSTO} from "../interfaces/IVeSTO.sol";

/// @title Voting Escrow for STO
contract Escrow is Ownable2Step {
    /*//////////////////////////////////////////////////////////////////////////
                                STATE VARIABLES
    //////////////////////////////////////////////////////////////////////////*/

    address public immutable STO;
    address public immutable veSTO;

    uint256 public lockDuration = 30 days;

    mapping(address => LockInfo[]) public lockInfo;

    /*//////////////////////////////////////////////////////////////////////////
                                    STRUCTS
```

```
//////////////////////////////////////////////////////////////*/

struct LockInfo {
    uint256 amount;
    uint256 unlockTime;
    bool claimed;
}

/*//////////////////////////////////////////////////////////////
                                EVENTS
//////////////////////////////////////////////////////////////*/
event Lock(address indexed owner, address indexed receiver, uint256 amount);
event Unlock(
    address indexed owner,
    address indexed receiver,
    uint256 amount
);
event Claim(address indexed receiver, uint256 amount);
event SetLockDuration(uint256 oldVal, uint256 newVal);

/*//////////////////////////////////////////////////////////////
                                CONSTRUCTOR
//////////////////////////////////////////////////////////////*/

constructor(address _sto, address _veSTO, address _admin) Ownable(_admin) {
    STO = _sto;
    veSTO = _veSTO;
}

/*//////////////////////////////////////////////////////////////
                                PERMISSIONLESS FUNCTIONS
//////////////////////////////////////////////////////////////*/

function lock(uint256 _amount) external returns (uint256 minted) {
    minted = _lockFor(_amount, msg.sender);
}

function lockFor(
    uint256 _amount,
    address _receiver
) external returns (uint256 minted) {
    minted = _lockFor(_amount, _receiver);
}

function unlock(uint256 _amount) external {
    _unlockFor(_amount, msg.sender);
}

function unlockFor(uint256 _amount, address _receiver) external {
    _unlockFor(_amount, _receiver);
}

function claim(uint256 _index) external {
    _claim(_index);
}

function claimBatch(uint256[] memory _indices) external {
    uint256 length = _indices.length;
    require(length != 0, "invalid indices");

    uint256 i;
    for (i; i < length; i++) {
        uint256 index = _indices[i];
        _claim(index);
    }
}

/*//////////////////////////////////////////////////////////////////////////////
                                VIEW FUNCTIONS
//////////////////////////////////////////////////////////////////////////////*/

function getUnlockInfo(
    address _user
) external view returns (LockInfo[] memory info) {
    info = lockInfo[_user];
}

function getUnlockInfoByIndex(
    address _user,
    uint256 _index
) external view returns (LockInfo memory info) {
    info = lockInfo[_user][_index];
}

/*//////////////////////////////////////////////////////////////////////////////
                                ADMIN FUNCTIONS
//////////////////////////////////////////////////////////////////////////////*/

function setLockDuration(uint256 _duration) external onlyOwner {
    emit SetLockDuration(lockDuration, _duration);
    lockDuration = _duration;
}

/*//////////////////////////////////////////////////////////////////////////////
                                INTERNAL FUNCTIONS
//////////////////////////////////////////////////////////////////////////////*/

function _lockFor(
    uint256 _amount,
    address _receiver
) internal returns (uint256 minted) {
    require(_amount > 0, "zero amount");
    require(_receiver != address(0), "zero address");

    minted = _amount;
    TransferHelper.safeTransferFrom(STO, msg.sender, address(this), minted);

    IVeSTO(veSTO).mint(_receiver, minted);

    emit Lock(msg.sender, _receiver, minted);
}

function _unlockFor(uint256 _amount, address _receiver) internal {
    require(_amount > 0, "zero amount");
    require(_receiver != address(0), "zero address");

    TransferHelper.safeTransferFrom(
        veSTO,
        msg.sender,
        address(this),
        _amount
    );

    LockInfo memory info;
    info.amount = _amount;
    info.unlockTime = block.timestamp + lockDuration;

    lockInfo[_receiver].push(info);

    emit Unlock(msg.sender, _receiver, _amount);
}

function _claim(uint256 _index) internal {
    require(_index < lockInfo[msg.sender].length, "invalid index");

    LockInfo storage info = lockInfo[msg.sender][_index];

    uint256 amount = info.amount;

    require(block.timestamp > info.unlockTime, "locked");
    require(!info.claimed, "claimed");

    info.claimed = true;

    IVeSTO(veSTO).burn(address(this), amount);
    TransferHelper.safeTransfer(STO, msg.sender, amount);
    emit Claim(msg.sender, amount);
}
}
```

## Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.