SLOWMIST

Smart Contract Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the Tomi on 2022.08.30. The following are the details and results of this smart contract security audit:

Token Name :

Tomi

File name and hash (SHA256) :

-7835397550881556526tomitoken.sol:

ec992fc84b50467aa181bba7a189e63987d376fbce48d605dad9d0f725143390

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	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed

NO.	Audit Items	Result
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002209010002

Audit Date : 2022.08.30 - 2022.09.01

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section and does not contain dark coin functions. The total amount of contract tokens can be changed. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following information:

1. Only the inititalizer role can call the initialize function once for initialization.
2. Only the nftContract can call mintThroughNft function to mint tokens to buyer, marketingWallet, and tomiWallet, and the nftContract is not in the audit scope.
3. Only the rewardPoolSuperComputer can call tomiMining function to mint tokens to rewardPoolSuperComputer and tomiWallet, and the rewardPoolSuperComputer is not in the audit scope. After communication with the project team, they expressed that the mintThroughNft function will be called by the Pioneer NFT Contract only which can be initialized once and can't be changed ever again to control mints at every Pioneer NFT sale. And the tomiMining function is controlled by their tomi Mining mechanism. There will be reward/Treasury Pool where the super computer will deposit its earnings, and the amount of money(USDT or ETH) it deposits in the reward pool, as it deposits the token tomiMining function will be called which will mint the amount calculated by the rewardPool to itself and a percentage to tomi wallet.

The rewards minted to rewardPoolSuperComputer will be then distributed to all the people who have bought TCP. This rewardPoolSuperComputer address and contract can only be initialized once and can be changed only by the DAO trustees. As after calling initialize the contract ownership will be transferred to GnosisSafe with trustees assigned.

The source code:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.0;
// SPDX-License-Identifier: GPL
//SlowMist// SafeMath security module is used, which is a recommended approach
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) {
      // Solidity only automatically asserts when dividing by 0
      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;
    }
}

/*
 * @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) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        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 {
        //SlowMist// This check is quite good in avoiding losing control of the
contract caused by user mistakes
        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);
    }
}


interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

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

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` 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 sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    /**
     * @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 Interface for the optional metadata functions from the BEP20 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.
     */
    function symbol() external view returns (string memory);

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

/**
 * @dev Required interface of an ERC721 compliant contract.
 */
interface IERC721 {
    /**
```

```
     * @dev To Check Whether Auction Ended or not`.
     */
    function hasAuctionEnded() external view returns (bool);
}

/**
* Tomi Super Computer rewards Pool interface
 */

interface IRewardsPool{
    function totalEarningDeposited() external view returns (uint256);
    function totalEarningMined() external view returns (uint256);
    function totalEarningDistributed() external view returns(uint256);
}

/**
 * @dev Implementation of the {IBEP20} 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 {BEP20}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/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 BEP20
 * 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 {BEP20-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;

    /**
     * @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}.
     */
    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);
        //SlowMist// The return value conforms to the EIP20 specification
        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);
        //SlowMist// The return value conforms to the EIP20 specification
        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);
        //SlowMist// The return value conforms to the EIP20 specification
        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");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        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;
        }
        _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;
    _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;
    }
```

```
        _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");
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to approve errors
        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 {}
}



contract Tomi is ERC20, Ownable {
    // owner will be a multiSig, trustee's of DAO will need to approve every change.
    using SafeMath for uint256;

    //  wallets
    address public marketingWallet;
    address public tomiWallet;

    IRewardsPool public rewardPoolSuperComputer;

    IERC721 public nftContract;

    bool private initialized = false;
    address inititalizer;

    uint256 public totalMined;


    // modifiers to limit contract functionality
    modifier canMint{
        require(emissions.mintAllowed, "Minting is not Allowed as of now.");
        require(_msgSender() == address(nftContract), "Only NFT Contract");
        _;
    }

    modifier canInitialize{
        require(initialized == false, "Can Only be Initialized Once");
        require(_msgSender() == inititalizer, "Only Inititalizer");
        _;
    }
```

```
     modifier miningAllowed{
        require(emissions.miningAllowed, "Mining is not Allowed as of now.");
        require(_msgSender() == address(rewardPoolSuperComputer), "Only Reward
Pool");

require(rewardPoolSuperComputer.totalEarningDeposited().sub(rewardPoolSuperComputer.t
otalEarningMined()) > 0, "Nothing to mine");
        _;
    }

    struct emissionCriteria{
        // tomi mints before auction first 2 weeks of NFT Minting
       uint256 beforeAuctionBuyer;
       uint256 beforeAuctionTomi;
       uint256 beforeAuctionMarketing;

        // tomi mints after two weeks // auction everyday of NFT Minting
       uint256 afterAuctionBuyer;
       uint256 afterAuctionTomi;
       uint256 afterAuctionMarketing;

       // booleans for checks of minting
       bool mintAllowed;

       // Mining Criteria and checks
       bool miningAllowed;
       uint8 poolPercentage;
       uint8 tomiPercentage;
    }

    emissionCriteria public emissions;

    // constructor
    constructor() ERC20("Tomi Token", "Tomi") {
         inititalizer = _msgSender();
    }
```

//SlowMist// Only the inititalizer role can call the initialize function once for initialization

```
    function initialize(IERC721 nftContract_, address marketingWallet_, address
tomiWallet_, IRewardsPool rewardPoolSuperComputer_) external canInitialize  {
        nftContract = nftContract_;
        marketingWallet = marketingWallet_;
        tomiWallet = tomiWallet_;
        rewardPoolSuperComputer = rewardPoolSuperComputer_;
        initialized = true;
        emissions = emissionCriteria(18000 ether, 39000 ether, 8000 ether, 18000
ether, 17000 ether, 0, true, true, 90, 10);
    }

    // DAO Voting Functions
    //SlowMist// Missing the event log and zero address check
    function updateMarketingWallet(address newAddress) external onlyOwner {
        marketingWallet = newAddress;
    }
    //SlowMist// Missing the event log and zero address check
    function updateTomiWallet(address newAddress) external onlyOwner {
        tomiWallet = newAddress;
    }
    //SlowMist// Missing the event log and zero address check
    function updateRewardPoolSuperComputer(IRewardsPool newAddress) external
onlyOwner {
        rewardPoolSuperComputer = newAddress;
    }
    //SlowMist// Missing the event log
    function updateEmissions(emissionCriteria calldata emissions_) external
onlyOwner{
            emissions = emissions_;
    }


    // Contract Functions
    function _transfer(address from, address to, uint256 amount) internal override {
        require(from != to, "Sending to yourself is disallowed");
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");
        require(amount > 0, "Transfer amount must be greater than zero");
        super._transfer(from, to, amount);
    }
    //SlowMist// Only the nftContract can call this function
    function mintThroughNft(address buyer) external canMint returns(bool){
        if(nftContract.hasAuctionEnded()){
            afterAuctionMint(buyer);
        }else{
            beforeAuctionMint(buyer);
        }
        return true;
    }
    function beforeAuctionMint(address buyer) internal {
        _mint(buyer, emissions.beforeAuctionBuyer);
        _mint(marketingWallet, emissions.beforeAuctionMarketing);
        _mint(tomiWallet, emissions.beforeAuctionTomi);
    }

    function afterAuctionMint(address buyer) internal {
        _mint(buyer, emissions.afterAuctionBuyer);
        _mint(tomiWallet, emissions.afterAuctionTomi);
        _mint(marketingWallet, emissions.afterAuctionMarketing);
    }

    //SlowMist// Only the rewardPoolSuperComputer can call this function
    function tomiMining(uint256 amount) miningAllowed external returns(bool) {
        totalMined = totalMined.add(amount);
        _mint(address(rewardPoolSuperComputer),
amount.mul(emissions.poolPercentage).div(100));
        _mint(tomiWallet, amount.mul(emissions.tomiPercentage).div(100));
        return true;
    }

}
```

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.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist