Smart Contract Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the ELToken on 2023.02.07. The following are the details and results of this smart contract security audit:

Token Name :

ELToken

The contract address :

https://etherscan.io/token/0x2781246fe707bb15cee3e5ea354e2154a2877b16

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	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
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002302090001

Audit Date : 2023.02.07 - 2023.02.09

Audit Team : SlowMist Security Team

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

During the audit, we found the following information:

1. The minter role can add new minters and minters can mint tokens arbitrarily through the mint function and there is no upper limit on the amount of tokens that can be minted that will lead to token inflation. After communication with the project team, they renounced the Minter role so that there is no minter can mint tokens anymore. The transaction is as follows: https://etherscan.io/tx/0x57d28410a4edd21b854c66c4067a6b1c287ac1fe9c5836c8e488070ae0014a9d

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2019-04-27
*/
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.5.0;

interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns
(bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns
(uint256);
```

```
    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract PauserRole {
    using Roles for Roles.Role;

    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    Roles.Role private _pausers;

    constructor () internal {
        _addPauser(msg.sender);
    }

    modifier onlyPauser() {
        require(isPauser(msg.sender));
        _;
    }

    function isPauser(address account) public view returns (bool) {
        return _pausers.has(account);
    }

    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function renouncePauser() public {
        _removePauser(msg.sender);
    }

    function _addPauser(address account) internal {
        _pausers.add(account);
        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);
        emit PauserRemoved(account);
    }
}

library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }
    function add(Role storage role, address account) internal {
        require(account != address(0));
        require(!has(role, account));

        role.bearer[account] = true;
    }

    function remove(Role storage role, address account) internal {
        require(account != address(0));
        require(has(role, account));

        role.bearer[account] = false;
    }

    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0));
        return role.bearer[account];
    }
}
//SlowMist// OpenZeppelin's SafeMath security module is used, which is a recommended
approach
library SafeMath {
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0);
        uint256 c = a / b;

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowed;

    uint256 private _totalSupply;

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    function allowance(address owner, address spender) public view returns (uint256)
{
        return _allowed[owner][spender];
    }

    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    function approve(address spender, uint256 value) public returns (bool) {
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to approve errors
        require(spender != address(0));

        _allowed[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    function transferFrom(address from, address to, uint256 value) public returns
(bool) {
        _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
        _transfer(from, to, value);
        emit Approval(from, msg.sender, _allowed[from][msg.sender]);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public returns
(bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = _allowed[msg.sender]
[spender].add(addedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public
returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = _allowed[msg.sender]
[spender].sub(subtractedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }

    function _transfer(address from, address to, uint256 value) internal {
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        require(to != address(0));

        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
        emit Transfer(from, to, value);
    }

    function _mint(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.add(value);
        _balances[account] = _balances[account].add(value);
        emit Transfer(address(0), account, value);
    }
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    function _burnFrom(address account, uint256 value) internal {
        _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(value);
        _burn(account, value);
        emit Approval(account, msg.sender, _allowed[account][msg.sender]);
    }
}

contract MinterRole {
    using Roles for Roles.Role;

    event MinterAdded(address indexed account);
    event MinterRemoved(address indexed account);

    Roles.Role private _minters;

    constructor () internal {
        _addMinter(msg.sender);
    }

    modifier onlyMinter() {
        require(isMinter(msg.sender));
        _;
    }

    function isMinter(address account) public view returns (bool) {
        return _minters.has(account);
    }
    //SlowMist// The Minter role can add new minter
    function addMinter(address account) public onlyMinter {
        _addMinter(account);
    }

    function renounceMinter() public {
        _removeMinter(msg.sender);
    }

    function _addMinter(address account) internal {
        _minters.add(account);
        emit MinterAdded(account);
    }
    function _removeMinter(address account) internal {
        _minters.remove(account);
        emit MinterRemoved(account);
    }
}

contract ERC20Mintable is ERC20, MinterRole {
    //SlowMist// The minter role can mint tokens arbitrarily through the mint
function and there is no upper limit on the amount of tokens
    function mint(address to, uint256 value) public onlyMinter returns (bool) {
        _mint(to, value);
        return true;
    }
}

contract ERC20Burnable is ERC20 {
    function burn(uint256 value) public {
        _burn(msg.sender, value);
    }
    //SlowMist// Because burnFrom() and transferFrom() share the allowed amount of
approve(), if the agent be evil, there is the possibility of malicious burn
    function burnFrom(address from, uint256 value) public {
        _burnFrom(from, value);
    }
}

contract Pausable is PauserRole {
    event Paused(address account);
    event Unpaused(address account);

    bool private _paused;

    constructor () internal {
        _paused = false;
    }

    function paused() public view returns (bool) {
        return _paused;
    }

    modifier whenNotPaused() {
        require(!_paused);
        _;
    }

    modifier whenPaused() {
        require(_paused);
        _;
    }
    //SlowMist// Suspending all transactions upon major abnormalities is a
recommended approach
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }

    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}

contract ERC20Pausable is ERC20, Pausable {
    function transfer(address to, uint256 value) public whenNotPaused returns (bool)
{
        return super.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) public
whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    function approve(address spender, uint256 value) public whenNotPaused returns
(bool) {
        return super.approve(spender, value);
    }

    function increaseAllowance(address spender, uint addedValue) public whenNotPaused
returns (bool success) {
        return super.increaseAllowance(spender, addedValue);
    }

    function decreaseAllowance(address spender, uint subtractedValue) public
whenNotPaused returns (bool success) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

contract Ownable {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed
newOwner);

    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

    function owner() public view returns (address) {
        return _owner;
    }

    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    function _transferOwnership(address newOwner) internal {
        //SlowMist// This check is quite good in avoiding losing control of the
contract caused by user mistakes
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
//SlowMist// Redundant contract
contract Migrations {
  address public owner;
  uint public last_completed_migration;

  constructor() public {
    owner = msg.sender;
  }

  modifier restricted() {
    if (msg.sender == owner) _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }

  function upgrade(address new_address) public restricted {
    Migrations upgraded = Migrations(new_address);
    upgraded.setCompleted(last_completed_migration);
  }
}

contract ELToken is ERC20Detailed, ERC20Burnable, ERC20Mintable, ERC20Pausable,
Ownable {
  constructor(string memory name, string memory symbol, uint8 decimals, uint256
totalSupply) ERC20Detailed(name, symbol, decimals) public {
    _mint(owner(), totalSupply * 10 ** uint(decimals));
    emit Transfer(address(0), msg.sender, totalSupply * 10 ** uint(decimals)); //
ERC20Basic.sol
  }
}
```

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.