Smart Contract Security Audit Report

The SlowMist Security Team received the ReapChain team's application for smart contract security audit of the REAP on July 28, 2020. The following are the details and results of this smart contract security audit:

Token name :

REAP

The Contract address :

0x1fc5EF0337AEA85C5f9198853a6E3A579a7A6987

Link address :

https://etherscan.io/address/0x1fc5EF0337AEA85C5f9198853a6E3A579a7A6987

The audit items and results :

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

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed

9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002007300001

Audit Date : July 30, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The total amount of contract tokens remains unchanged. The pauser role can block the `_user` transfer operation through the pauseUser function. OpenZeppelin's SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

IERC20.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.6.6;

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
interface IERC20 {
```

```
  function totalSupply() external view returns (uint256);

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

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

  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);

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

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

SafeMath.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.6.6;


/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */

//SlowMist// OpenZeppelin's SafeMath security Module is used, which is a recommend approach

library SafeMath {
```

```
/**
* @dev Multiplies two numbers, reverts on 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-solidity/pull/522
  if (a == 0) {
    return 0;
  }

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

  return c;
}

/**
* @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
*/
function div(uint256 a, uint256 b) internal pure returns (uint256) {
  require(b > 0); // Solidity only automatically asserts when dividing by 0
  uint256 c = a / b;
  // assert(a == b * c + a % b); // There is no case in which this doesn't hold

  return c;
}

/**
* @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
*/
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
  require(b <= a);
  uint256 c = a - b;

  return c;
}

/**
* @dev Adds two numbers, reverts on overflow.
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two numbers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}
```

ERC20.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.6.6;

import "./IERC20.sol";
import "./SafeMath.sol";

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
 * Originally based on code by FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;
```

```
/**
 * @dev Total number of tokens in existence
 */
function totalSupply() public view override returns (uint256) {
    return _totalSupply;
}

/**
 * @dev Gets the balance of the specified address.
 * @param owner The address to query the balance of.
 * @return An uint256 representing the amount owned by the passed address.
 */
function balanceOf(address owner) public view override returns (uint256) {
    return _balances[owner];
}

/**
  * @dev Function to check the amount of tokens that an owner allowed to a spender.
  * @param owner address The address which owns the funds.
  * @param spender address The address which will spend the funds.
  * @return A uint256 specifying the amount of tokens still available for the spender.
  */
function allowance(
    address owner,
    address spender
 )
    public
    view
    override
    returns (uint256)
{
    return _allowed[owner][spender];
}

/**
 * @dev Transfer token for a specified address
 * @param to The address to transfer to.
 * @param value The amount to be transferred.
 */
function transfer(address to, uint256 value) public virtual override returns (bool) {
    require(value <= _balances[msg.sender]);
```

```
    require(to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

    _balances[msg.sender] = _balances[msg.sender].sub(value);
    _balances[to] = _balances[to].add(value);
    emit Transfer(msg.sender, to, value);

    return true; //SlowMist// The return value conforms to the EIP20 specification
  }

  /**
   * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
   * 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
   * @param spender The address which will spend the funds.
   * @param value The amount of tokens to be spent.
   */
  function approve(address spender, uint256 value) public virtual override returns (bool) {
    require(spender != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to approve errors

    _allowed[msg.sender][spender] = value;
    emit Approval(msg.sender, spender, value);

    return true; //SlowMist// The return value conforms to the EIP20 specification
  }

  /**
   * @dev Transfer tokens from one address to another
   * @param from address The address which you want to send tokens from
   * @param to address The address which you want to transfer to
   * @param value uint256 the amount of tokens to be transferred
   */
  function transferFrom(
    address from,
    address to,
```

```
    uint256 value
)
    public
    virtual
    override
    returns (bool)
{
    require(value <= _balances[from]);
    require(value <= _allowed[from][msg.sender]);

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


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

    return true; //SlowMist// The return value conforms to the EIP20 specification

}

/**
 * @dev Increase the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed_[_spender] == 0. To increment
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAC Token.sol
 * @param spender The address which will spend the funds.
 * @param addedValue The amount of tokens to increase the allowance by.
 */
function increaseAllowance(
    address spender,
    uint256 addedValue
)
    public
    virtual
    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;
}

/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed_[_spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param spender The address which will spend the funds.
 * @param subtractedValue The amount of tokens to decrease the allowance by.
 */
function decreaseAllowance(
    address spender,
    uint256 subtractedValue
)
    public
    virtual
    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;
}

/**
 * @dev Internal function that mints an amount of the token and assigns it to
 * an account. This encapsulates the modification of balances such that the
 * proper events are emitted.
 * @param account The account that will receive the created tokens.
 * @param amount The amount that will be created.
 */
function _mint(address account, uint256 amount) internal {
    require(account != address(0));
    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
```

```
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param amount The amount that will be burnt.
     */
    function _burn(address account, uint256 amount) internal {
        require(account != address(0));
        require(amount <= _balances[account]);

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

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses the
     * internal burn function.
     * @param account The account whose tokens will be burnt.
     * @param amount The amount that will be burnt.
     */
    //SlowMist// It's redundant code
    function _burnFrom(address account, uint256 amount) internal {
        require(amount <= _allowed[account][msg.sender]);

        // Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be accepted,
        // this function needs to emit an event with the updated approval.
        _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(
            amount);
        _burn(account, amount);
    }
}
```

Roles.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.6.6;
```

```
/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev give an account access to this role
     */
    function add(Role storage role, address account) internal {
        require(account != address(0));
        role.bearer[account] = true;
    }

    /**
     * @dev remove an account's access to this role
     */
    function remove(Role storage role, address account) internal {
        require(account != address(0));
        role.bearer[account] = false;
    }

    /**
     * @dev check if an account has this role
     * @return bool
     */
    function has(Role storage role, address account)
        internal
        view
        returns (bool)
    {
        require(account != address(0));
        return role.bearer[account];
    }
}
```

PauserRole.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.6.6;

import "./Roles.sol";

contract PauserRole {
    using Roles for Roles.Role;

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

    Roles.Role private pausers;

    constructor() public {
        pausers.add(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 {
        pausers.add(account);
        emit PauserAdded(account);
    }

    function renouncePauser() public {
        pausers.remove(msg.sender);
    }

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

Pausable.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.6.6;

import "./PauserRole.sol";

/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is PauserRole {
    event Paused();
    event Unpaused();

    bool private _paused = false;

    /**
     * @return true if the contract is paused, false otherwise.
     */
    function paused() public view returns(bool) {
        return _paused;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     */
    modifier whenNotPaused() {
        require(!_paused);
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     */
    modifier whenPaused() {
        require(_paused);
        _;
    }

    /**
     * @dev called by the owner to pause, triggers stopped state
```

```
    */
  //SlowMist// Suspending all transactions upon major abnormalities is a recommended approach
  function pause() public onlyPauser whenNotPaused {
      _paused = true;
      emit Paused();
  }

  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  function unpause() public onlyPauser whenPaused {
      _paused = false;
      emit Unpaused();
  }
}
```

PausableUser.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.6.6;

import "./PauserRole.sol";

/**
 * @title PausableUser
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract PausableUser is PauserRole {
  event PausedUser(address _user);
  event UnpausedUser(address _user);

  mapping (address => bool) private _pausedUser;

  /**
   * @return true if the address is paused, false otherwise.
   */
  function pausedUser(address _user) public view returns(bool) {
      return _pausedUser[_user];
  }
```

```
    /**
     * @dev Modifier to make a function callable only when the address is not paused.
     */
    modifier whenNotPausedUser(address _user) {
        require(!_pausedUser[_user]);
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the address is paused.
     */
    modifier whenPausedUser(address _user) {
        require(_pausedUser[_user]);
        _;
    }

    /**
     * @dev called by the owner to pause user, triggers stopped state
     */
    //SlowMist// The pauser role can block the _user transfer operation through the pauseUser function
    function pauseUser(address _user) public onlyPauser whenNotPausedUser(_user) {
        _pausedUser[_user] = true;
        emit PausedUser(_user);
    }

    /**
     * @dev called by the owner to unpause user, returns to normal state
     */
    function unpauseUser(address _user) public onlyPauser whenPausedUser(_user) {
        _pausedUser[_user] = false;
        emit UnpausedUser(_user);
    }
}
```

ERC20Pausable.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.6.6;
```

```
import "./ERC20.sol";
import "./Pausable.sol";
import "./PausableUser.sol";

/**
 * @title Pausable token
 * @dev ERC20 modified with pausable transfers.
 **/
contract ERC20Pausable is ERC20, Pausable, PausableUser {

  function transfer(
    address to,
    uint256 value
  )
    public
    override
    whenNotPaused
    whenNotPausedUser(msg.sender)
    returns (bool)
  {
    return super.transfer(to, value);
  }

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

  function approve(
    address spender,
    uint256 value
  )
```

```
    public
    override
    whenNotPaused
    whenNotPausedUser(msg.sender)
    returns (bool)
  {
    return super.approve(spender, value);
  }

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

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

ReapChain.sol:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.6.6;
```

```
import "./ERC20Pausable.sol";

/**
 * @title Pausable token
 * @dev ERC20 modified with pausable transfers.
 **/
contract ReapChain is ERC20Pausable {
    string public constant name = "ReapChain";
    uint8 public constant decimals = 18;
    string public constant symbol = "REAP";

    constructor() public {
        _mint(msg.sender, 4900000000 * 1000000000000000000);
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account