Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the SupervisedMultiTimelock on 2022.02.07. The following are the details and results of this smart contract security audit:

Token Name :

SupervisedMultiTimelock

The contract address :

https://github.com/octopus-network/oct-token-eth/blob/main/contracts/SupervisedMultiTimelock.sol

commit: 6cf195c5d39feed40177b6267a1da4ed07d36ecc

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

NO.	Audit Items	Result
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed

Audit Result : Passed

Audit Number : 0X002202090001

Audit Date : 2022.02.07 - 2022.02.09

Audit Team : SlowMist Security Team

Summary conclusion : This is a timelock contract. The total amount of contract tokens can be changed, the owner can add the beneficiary address to get benefits. The contract does not have the Overflow and the Race Conditions issue.

During the audit, we found the following information:

1. The owner role can delete the beneficiary address and the beneficiary address can not withdraw the benefit from this contract anymore.
2. The owner role can withdraw the remaining tokens in this contract.

The source code:

```
// SPDX-License-Identifier: GPL-3.0
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

struct Benefit {
    uint256 totalAmount;
```

```
    uint256 releaseStartTime;
    uint256 releaseEndTime;
    uint256 withdrawnAmount;
}

/**
 * @dev A token holder contract that will allow a beneficiary to withdraw the
 * tokens after a given release time.
 */
contract SupervisedMultiTimelock is Ownable {
    using SafeERC20 for IERC20;
    // Seconds of a day
    uint256 private constant SECONDS_OF_A_DAY = 86400;
    // The OctToken contract
    IERC20 private immutable _token;

    mapping(address => Benefit) _beneficiaries;

    event BenefitIsIssued(
        address indexed beneficiary,
        uint256 amount,
        uint256 releaseStartTime,
        uint256 daysOfTimelock
    );
    event BenefitIsWithdrawn(address indexed beneficiary, uint256 amount);
    event BenefitIsTerminated(address indexed beneficiary);

    constructor(IERC20 token_) {
        _token = token_;
    }

    /**
     * @return the token being held.
     */
    function token() public view returns (IERC20) {
        return _token;
    }

    /**
     * @return the issued benefit of a beneficiary
     */
    function issuedBenefitOf(address addr) public view returns (uint256) {
        return _beneficiaries[addr].totalAmount;
    }
```

```
/**
 * @return the amount which can be withdrawn by a beneficiary at the moment
 */
function releasedAmountOf(address addr) public view returns (uint256) {
    Benefit memory benefit = _beneficiaries[addr];
    if (benefit.totalAmount == 0) return 0;
    if (block.timestamp <= benefit.releaseStartTime) return 0;
    if (block.timestamp > benefit.releaseEndTime) {
        return benefit.totalAmount;
    }
    uint256 passedDays = (block.timestamp - benefit.releaseStartTime) /
        SECONDS_OF_A_DAY;
    uint256 totalDays = (benefit.releaseEndTime -
        benefit.releaseStartTime) / SECONDS_OF_A_DAY;
    return (benefit.totalAmount * passedDays) / totalDays;
}

/**
 * @return the unreleased amount of a beneficiary at the moment
 */
function unreleasedAmountOf(address addr) public view returns (uint256) {
    Benefit memory benefit = _beneficiaries[addr];
    if (benefit.totalAmount == 0) return 0;
    return benefit.totalAmount - releasedAmountOf(addr);
}

/**
 * @return the withdrawn amount of a beneficiary at the moment
 */
function withdrawnAmountOf(address addr) public view returns (uint256) {
    return _beneficiaries[addr].withdrawnAmount;
}

/**
 * @notice Issue a certain amount benefit to a beneficiary.
 * An address of a beneficiary can only be used once.
 */
function issueBenefitTo(
    address addr,
    uint256 totalAmount_,
    uint256 releaseStartTime_,
    uint256 daysOfTimelock_
) public onlyOwner {
```

```
        require(
            issuedBenefitOf(addr) == 0,
            "SupervisedMultiTimelock: the address is already a beneficiary."
        );
        releaseStartTime_ -= (releaseStartTime_ % SECONDS_OF_A_DAY);
        uint256 releaseEndTime = releaseStartTime_ +
            daysOfTimelock_ *
            SECONDS_OF_A_DAY;
        require(
            releaseEndTime > block.timestamp,
            "SupervisedMultiTimelock: release end time is before current time."
        );
        require(
            totalAmount_ > 0,
            "SupervisedMultiTimelock: the total amount should be greater than 0."
        );
        _beneficiaries[addr] = Benefit({
            totalAmount: totalAmount_,
            releaseStartTime: releaseStartTime_,
            releaseEndTime: releaseEndTime,
            withdrawnAmount: 0
        });

        emit BenefitIsIssued(
            addr,
            totalAmount_,
            releaseStartTime_,
            daysOfTimelock_
        );
    }

    /**
     * @notice Withdraws benefit of a beneficiary.
     * This function can be called by any account.
     */
    function withdrawBenefitOf(address addr) public {
        require(
            releasedAmountOf(addr) > withdrawnAmountOf(addr),
            "SupervisedMultiTimelock: no more benefit to withdraw."
        );
        uint256 amount = releasedAmountOf(addr) - withdrawnAmountOf(addr);
        require(
            token().balanceOf(address(this)) >= amount,
            "SupervisedMultiTimelock: deposited amount is not enough."
```

```
        );

        _beneficiaries[addr].withdrawnAmount += amount;
        token().safeTransfer(addr, amount);

        emit BenefitIsWithdrawn(addr, amount);
    }

    /**
     * @notice Remove a certain beneficiary from this contract.
     * The removed beneficiary can not withdraw benefit from this contract any more.
     */
    //SlowMist// The owner role can delete the beneficiary address and the
beneficiary address can not withdraw the benefit from this contract any more
    function terminateBenefitOf(address addr) public onlyOwner {
        delete _beneficiaries[addr];

        emit BenefitIsTerminated(addr);
    }

    /**
     * @notice Withdraw a certain amount of remaining benefit to the owner.
     */
    //SlowMist// The owner role can withdraw the remaining tokens in this contract
    function withdrawRemainingBenefit(uint256 amount) public onlyOwner {
        uint256 remainingAmount = token().balanceOf(address(this));
        require(
            amount <= remainingAmount,
            "SupervisedMultiTimelock: deposited amount is not enough."
        );
        token().safeTransfer(owner(), amount);

        emit BenefitIsWithdrawn(owner(), 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist