智能合约安全审计报告

[2021]

慢雾安全团队于2021.08.10，收到Octopus Network团队对OctToken智能合约安全审计的申请，如下为本次智能合约安全审计细节及结果：

Token 名称：

OctToken

合约地址：

https://github.com/octopus-network/oct-token-eth/tree/main/contracts

Commit：

5388d6d3b184b2ccffcccc2d2acf2b7888aeabe1

本次审计项及结果：

（其他未知安全漏洞不包含在本次审计责任范围）

序号	审计类别	审计结果
1	重放攻击	通过
2	拒绝服务攻击	通过
3	条件竞争攻击	通过
4	权限控制攻击	通过
5	整数上溢/下溢攻击	通过
6	Gas优化设计	通过
7	业务逻辑缺陷审计	通过
8	未声明的存储指针	通过
9	算术精度误差	通过
10	假充值漏洞	通过

序号	审计类别	审计结果
11	恶意 Event 事件审计	通过
12	变量声明及作用域审计	通过
13	安全设计审计	通过

审计结果：通过

审计编号：0x002108130002

审计日期：2021.08.10 - 2021.08.13

审计团队：SlowMist Security Team

备注：审计意见及建议见代码注释 //SlowMist//.....

总结： 此为代币 (token) 合约，包含时间锁 (Timelock) 部分。合约的代币总量不可变。使用了 SafeMath 安全模块，值得称赞的做法。合约不存在溢出、条件竞争问题。合约存在权限过大的风险问题。

在审计过程中，我们发现如下信息：

1. 只有 owner 角色可以通过 benefit 函数向指定带有 supervised 标志的受益人地址添加余额金额。
2. 只有 owner 角色可以通过 decreaseBenefitOf 函数减少受益人地址的 unreleased supervised 数量。

合约源代码如下：

OctToken.sol

```
// SPDX-License-Identifier: GPL-3.0
//SlowMist// 合约不存在溢出、条件竞争问题
pragma solidity ^0.8.0;

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

contract OctToken is ERC20, Ownable {
    // Total supply: 100 million
    uint256 private constant TOTAL_SUPPLY = 100000000;
```

```
    /**
     * @dev Initializes the contract, mint total supply to the deployer (owner).
     */
    constructor() ERC20("OctToken", "OCT") {
        _mint(msg.sender, TOTAL_SUPPLY * 10**(uint256(decimals())));
    }
}
```

OctFoundationTimelock.sol

```
// SPDX-License-Identifier: GPL-3.0
//SlowMist// 合约不存在溢出、条件竞争问题
pragma solidity ^0.8.0;

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

// The storage data of a beneficiary
//
// Because the smart contract can NOT automatically execute over time,
// the value of 'unreleasedBalance', 'unreleasedSupervisedBalance' and
'releasedBalance'
// will be updated ONLY when 'unreleasedBalance' or 'unreleasedSupervisedBalance'
// need to be modified during release period (from EARLIEST_RELEASE_START_TIME to
RELEASE_END_TIME)
// by calling function '_benefit(address, amount, supervised)' or
'decreaseBenefitOf(address, amount)'
struct Beneficiary {
    // The amount of unreleased balance of the beneficiary.
    //
    // This value may NOT be equal to the actual unreleased balance,
    // call function 'unreleasedBalanceOf(address)' to get actual value.
    uint256 unreleasedBalance;
    // The amount of unreleased supervised balance of the beneficiary.
    //
    // This value may NOT be equal to the actual unreleased supervised balance,
    // call function 'unreleasedSupervisedBalanceOf(address)' to get actual value.
    uint256 unreleasedSupervisedBalance;
    // The amount of released balance of the beneficiary.
    //
    // This value may NOT be equal to the actual total released balance,
    // call function 'releasedBalanceOf(address)' to get actual value.
    uint256 releasedBalance;
    // The amount of withdrawed balance of the beneficiary.
```

```
    //
    // This value will be updated on each withdraw operation.
    uint256 withdrawedBalance;
    // The start time when the beneficiary can withdraw held tokens.
    //
    // This value will be updated ONLY when 'unreleasedBalance' or
'unreleasedSupervisedBalance'
    // is changed during release period (from EARLIEST_RELEASE_START_TIME to
RELEASE_END_TIME)
    // for recalculating the time lock amount of held balance of beneficiary.
    uint256 releaseStartTime;
}

/**
 * @dev A token holder contract that will allow a beneficiary to withdraw the
 * tokens after a given release time.
 */
contract OctFoundationTimelock is Ownable {
    using SafeERC20 for IERC20;

    // Seconds of a day
    uint256 private constant SECONDS_OF_A_DAY = 86400;

    // The earliest timestamp of token release period (2021/09/01 00:00:00 GMT).
    //
    // Before this time NO ONE can withdraw any token from this contract.
    uint256 private constant EARLIEST_RELEASE_START_TIME = 1630454400;

    // The end timestamp of token release period (2024/09/01 00:00:00 GMT).
    //
    // After this time, ANY ONE can withdraw any amount of tokens they held.
    uint256 private constant RELEASE_END_TIME = 1725148800;

    // The OctToken contract
    IERC20 private immutable _token;

    // Map of all beneficiaries
    mapping(address => Beneficiary) private _beneficiaries;

    event BenefitAdded(
        address indexed beneficiary,
        uint256 amount,
        bool supervised
    );
    event BenefitReduced(address indexed beneficiary, uint256 amount);
```

```
    event BenefitTransfered(
        address indexed from,
        address indexed to,
        uint256 amount
    );
    event BenefitWithdrawed(address indexed beneficiary, uint256 amount);

    constructor(IERC20 token_) {
        _token = token_;
    }

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

    /**
     * @return the (supervised) balance to release for the given beneficiary at the
moment
     */
    function _balanceToReleaseTo(address addr, bool supervised)
        private
        view
        returns (uint256)
    {
        Beneficiary memory beneficiary = _beneficiaries[addr];
        if (block.timestamp <= beneficiary.releaseStartTime) return 0;
        if (block.timestamp > RELEASE_END_TIME) {
            if (supervised) return beneficiary.unreleasedSupervisedBalance;
            else return beneficiary.unreleasedBalance;
        }
        uint256 passedDays = (block.timestamp - beneficiary.releaseStartTime) /
            SECONDS_OF_A_DAY;
        uint256 totalDays = (RELEASE_END_TIME - beneficiary.releaseStartTime) /
            SECONDS_OF_A_DAY;
        if (supervised)
            return
                (beneficiary.unreleasedSupervisedBalance * passedDays) /
                totalDays;
        else return (beneficiary.unreleasedBalance * passedDays) / totalDays;
    }

    /**
```

```
     * @return the unreleased balance of the given beneficiary at the moment
     */
    function unreleasedBalanceOf(address addr) public view returns (uint256) {
        return
            _beneficiaries[addr].unreleasedBalance -
            _balanceToReleaseTo(addr, false);
    }

    /**
     * @return the unreleased supervised balance of the given beneficiary at the
moment
     */
    function unreleasedSupervisedBalanceOf(address addr)
        public
        view
        returns (uint256)
    {
        return
            _beneficiaries[addr].unreleasedSupervisedBalance -
            _balanceToReleaseTo(addr, true);
    }

    /**
     * @return the balance which can be withdrawed by the given beneficiary at the
moment
     */
    function releasedBalanceOf(address addr) public view returns (uint256) {
        return
            _beneficiaries[addr].releasedBalance +
            _balanceToReleaseTo(addr, false) +
            _balanceToReleaseTo(addr, true);
    }

    /**
     * @return the withdrawed balance of the given beneficiary at the moment
     */
    function withdrawedBalanceOf(address addr) public view returns (uint256) {
        return _beneficiaries[addr].withdrawedBalance;
    }

    /**
     * @notice Withdraws tokens to beneficiary
     */
    function withdraw(uint256 amount) public {
        uint256 withdrawedBalance = _beneficiaries[_msgSender()]
```

```
            .withdrawedBalance;
        require(
            releasedBalanceOf(_msgSender()) - withdrawedBalance >= amount,
            "OctFoundationTimelock: withdraw amount exceeds available released
balance"
        );
        require(
            token().balanceOf(address(this)) >= amount,
            "OctFoundationTimelock: deposited amount is not enough"
        );

        _beneficiaries[_msgSender()].withdrawedBalance =
            withdrawedBalance +
            amount;

        token().safeTransfer(_msgSender(), amount);

        emit BenefitWithdrawed(_msgSender(), amount);
    }

    /**
     * @notice Add amount of balance to the given beneficiary (address), with a flag
of supervised.
     */
    function _benefit(
        address addr,
        uint256 amount,
        bool supervised
    ) private {
        Beneficiary storage beneficiary = _beneficiaries[addr];
        if (block.timestamp < EARLIEST_RELEASE_START_TIME) {
            if (supervised) {
                beneficiary.unreleasedSupervisedBalance += amount;
            } else {
                beneficiary.unreleasedBalance += amount;
            }
            beneficiary.releaseStartTime = EARLIEST_RELEASE_START_TIME;
        } else {
            beneficiary.releasedBalance = releasedBalanceOf(addr);
            if (supervised) {
                beneficiary.unreleasedSupervisedBalance =
                    unreleasedSupervisedBalanceOf(addr) +
                    amount;
                beneficiary.unreleasedBalance = unreleasedBalanceOf(addr);
            } else {
```

```
                beneficiary
                    .unreleasedSupervisedBalance = unreleasedSupervisedBalanceOf(
                    addr
                );
                beneficiary.unreleasedBalance =
                    unreleasedBalanceOf(addr) +
                    amount;
            }
            beneficiary.releaseStartTime =
                block.timestamp -
                (block.timestamp % SECONDS_OF_A_DAY);
        }
    }

    /**
     * @notice Add amount of balance to the given beneficiary (address), with a flag
of supervised, which can ONLY be called by the owner.
     */
    //SlowMist// 只有 owner 角色可以通过 benefit 函数向指定带有 supervised 标志的受益人地址添加余额
金额
    function benefit(
        address addr,
        uint256 amount,
        bool supervised
    ) public onlyOwner {
        _benefit(addr, amount, supervised);

        emit BenefitAdded(addr, amount, supervised);
    }

    /**
     * @notice Transfer amount of unreleased balance of the caller to another account
(address).
     */
    function transferUnreleasedBalance(
        address addr,
        uint256 amount,
        bytes32 msgHash,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public {
        require(
            unreleasedBalanceOf(_msgSender()) >= amount,
            "OctFoundationTimelock: transfer amount exceeds unreleased balance"
```

```
        );
        require(
            ecrecover(msgHash, v, r, s) == addr,
            "OctFoundationTimelock: beneficiary MUST be an EOA"
        );
        Beneficiary storage beneficiary = _beneficiaries[_msgSender()];
        if (block.timestamp < EARLIEST_RELEASE_START_TIME) {
            beneficiary.unreleasedBalance -= amount;
            beneficiary.releaseStartTime = EARLIEST_RELEASE_START_TIME;
        } else {
            beneficiary.releasedBalance = releasedBalanceOf(_msgSender());
            beneficiary.unreleasedBalance =
                unreleasedBalanceOf(_msgSender()) -
                amount;
            beneficiary
                .unreleasedSupervisedBalance = unreleasedSupervisedBalanceOf(
                _msgSender()
            );
            beneficiary.releaseStartTime =
                block.timestamp -
                (block.timestamp % SECONDS_OF_A_DAY);
        }
        _benefit(addr, amount, false);

        emit BenefitTransfered(_msgSender(), addr, amount);
    }

    /**
     * @notice Decrease amount of unreleased supervised balance of a beneficiary
(address), which can ONLY be called by the owner.
     */
    //SlowMist// 只有 owner 角色可以通过 decreaseBenefitOf 函数减少受益人地址的 unreleased
supervised 数量
    function decreaseBenefitOf(address addr, uint256 amount) public onlyOwner {
        require(
            unreleasedSupervisedBalanceOf(addr) >= amount,
            "OctFoundationTimelock: decrease amount exceeds unreleased supervised
balance"
        );
        Beneficiary storage beneficiary = _beneficiaries[addr];
        if (block.timestamp < EARLIEST_RELEASE_START_TIME) {
            beneficiary.unreleasedSupervisedBalance -= amount;
            beneficiary.releaseStartTime = EARLIEST_RELEASE_START_TIME;
        } else {
            beneficiary.releasedBalance = releasedBalanceOf(addr);
```

```
            beneficiary.unreleasedBalance = unreleasedBalanceOf(addr);
            beneficiary.unreleasedSupervisedBalance =
                unreleasedSupervisedBalanceOf(addr) -
                amount;
            beneficiary.releaseStartTime =
                block.timestamp -
                (block.timestamp % SECONDS_OF_A_DAY);
        }

        emit BenefitReduced(addr, amount);
    }
}
```

声明

厦门慢雾科技有限公司(下文简称 ”慢雾”) 仅就本报告出具前项目方已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后项目方发生或存在的未知漏洞及安全事件，慢雾无法判断其安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向慢雾提供的文件和资料(简称“已提供资料”)。慢雾假设: 已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，慢雾对由此而导致的损失和不利影响不承担任何责任，慢雾仅对该项目的安全情况进行约定内的安全审计并出具了本报告，慢雾不对该项目背景及其他情况进行负责。

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号