BEOSIN
Blockchain Security

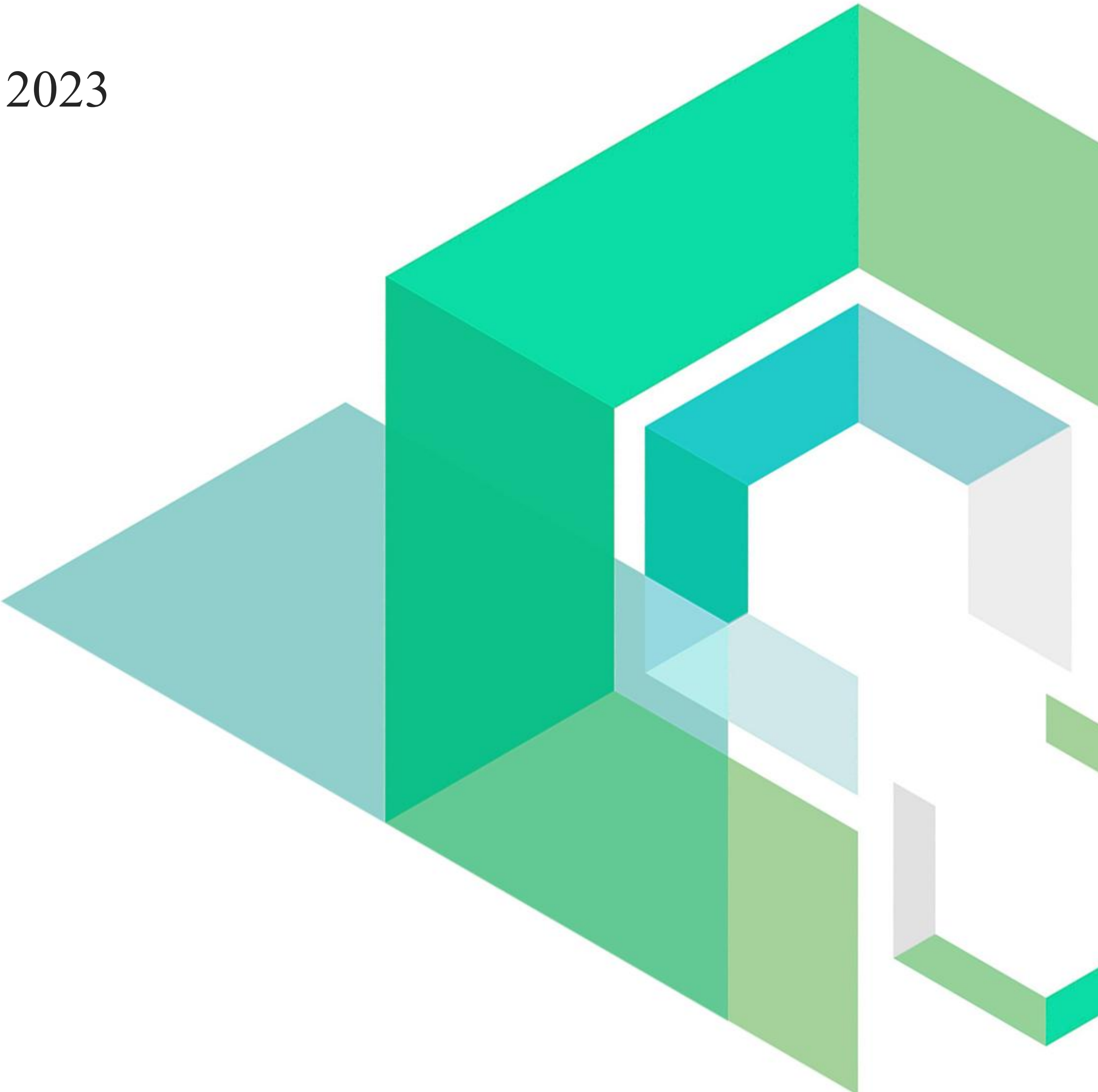
Otter-Airdrop

Smart Contract Security Audit

V1.0

No. 202304241330

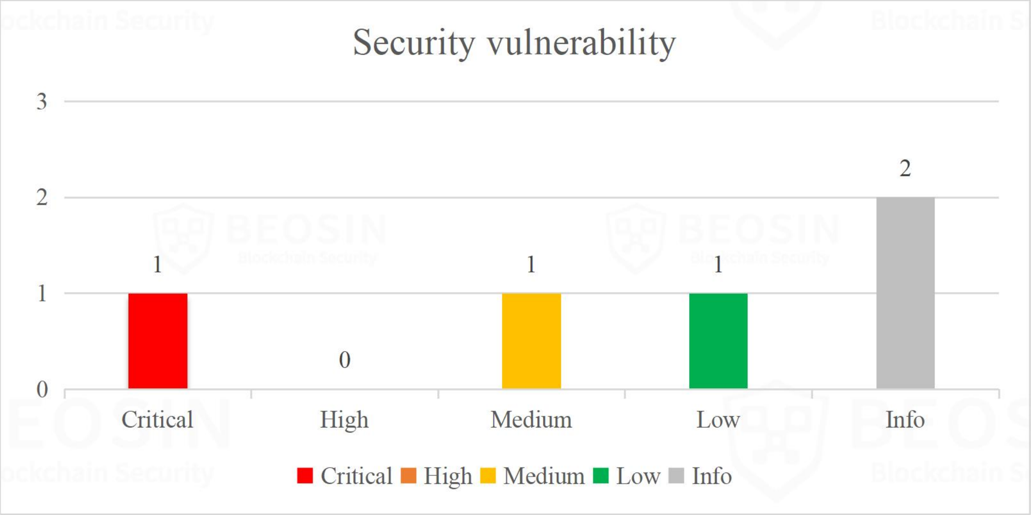
Apr 24th, 2023



Contents

Summary of Audit Results

After auditing, 1 Critical, 1 Medium, 1 Low and 2 Info risk items were identified in the Otter-Airdrop project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



[*]**Notes:**

- **Risk Description:**

1. If the funds are increased in the 12th month or after the 12th month of the current month, the increased funds will be locked in the contract and cannot be withdrawn.

- **Project Description:**

1. Business overview

The Otter-Airdrop project is an airdrop contract with a total of 90 million pieces. There are three roles involved: foundation role, board role and VIP role.

The foundation will release 30 million pieces in two stages: 15 million pieces in the first stage and 15 million pieces in the second stage. The first stage and the second stage will be separated by 90 days. The first member of the foundation will be set up as a beneficiary of the foundation. Only foundation beneficiary users can withdraw foundation airdrops.

The VIP will invest 60 million, which will be distributed to VIP users for 12 months, 5 million per month, and dividends will be distributed according to the proportion held by each VIP user, and the total ratio will not exceed 100.

This project allows the adjustment of VIP dividend members and corresponding ratios in the next month and subsequent months. Any user can initiate a proposal to change VIP members and adjust the corresponding dividend ratio. Signed by foundation members and board members. When the number of proposal signatures is equal to 3, it means that the proposal is passed. Passed proposals can be executed in the execute proposal function. The proposal time lock is 48 hours long, that is, the proposal can be executed 48 hours after it is passed.

1 Overview

1.1 Project Overview

Project Name	Otter-Airdrop
Platform	EVM Compatible Chains
Audit scope	https://github.com/treasurenetprotocol/contracts/blob/main/Airdrop/Airdrop.sol
Commit Hash	c2efb7167762d55b3ea3ea76d7089b876012307c c070cf2bd468bb747c477378d7073896aac79885 8f1a7b367d99dea8b495d2e7b406b0294f58f317 fd4c08088bbbb7056d387d8c0ed698a28329e746 d3ff9d2c7de398006a77a0f5f9d5fafdcb2f08e8

1.2 Audit Overview

Audit work duration: Apr 14, 2023 – Apr 24, 2023

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
Otter-Airdrop-1	Proposal arbitrary execution vulnerability	Critical	Fixed
Otter-Airdrop-2	Proposal reuse vulnerability	Medium	Fixed
Otter-Airdrop-3	Contract cannot withdraw funds	Low	Acknowledged
Otter-Airdrop-4	The management role operation did not trigger an event	Info	Fixed
Otter-Airdrop-5	Naming convention issue	Info	Fixed

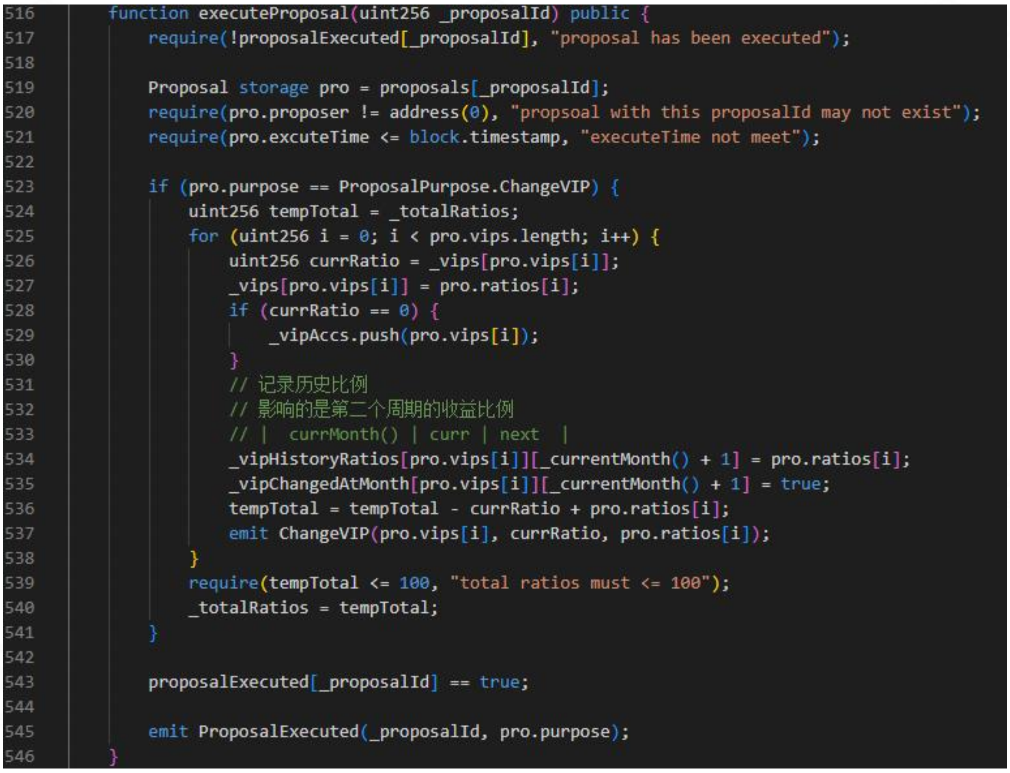
Status Notes:

1. Otter-Airdrop-3 is unfixed, the increased funds will be locked in the contract and cannot be withdrawn when the 12th month or after the 12th month.

Finding Details:

[Otter-Airdrop-1] Proposal arbitrary execution vulnerability

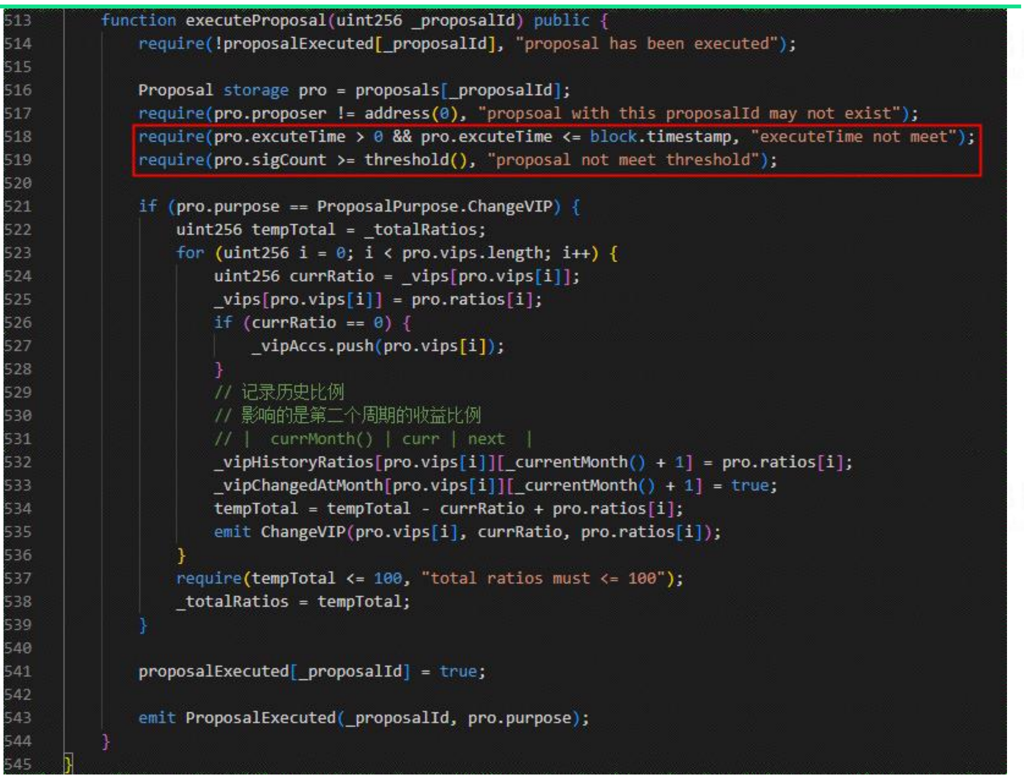
Severity Level	Critical
Type	Business Security
Lines	Airdrop.sol #L516-546
Description	Proposal execution does not verify that the proposal is approved, resulting in anyone being able to execute their own constructed proposal. Anyone who sets himself as a VIP role and distributes airdrop rewards leads to unexpected loss of contract funds.



```
516     function executeProposal(uint256 _proposalId) public {
517         require(!proposalExecuted[_proposalId], "proposal has been executed");
518 
519         Proposal storage pro = proposals[_proposalId];
520         require(pro.proposer != address(0), "propsoal with this proposalId may not exist");
521         require(pro.excuteTime <= block.timestamp, "executeTime not meet");
522 
523         if (pro.purpose == ProposalPurpose.ChangeVIP) {
524             uint256 tempTotal = _totalRatios;
525             for (uint256 i = 0; i < pro.vips.length; i++) {
526                 uint256 currRatio = _vips[pro.vips[i]];
527                 _vips[pro.vips[i]] = pro.ratios[i];
528                 if (currRatio == 0) {
529                     _vipAccs.push(pro.vips[i]);
530                 }
531                 // 记录历史比例
532                 // 影响的是第二个周期的收益比例
533                 // |  currMonth() | curr | next  |
534                 _vipHistoryRatios[pro.vips[i]][_currentMonth() + 1] = pro.ratios[i];
535                 _vipChangedAtMonth[pro.vips[i]][_currentMonth() + 1] = true;
536                 tempTotal = tempTotal - currRatio + pro.ratios[i];
537                 emit ChangeVIP(pro.vips[i], currRatio, pro.ratios[i]);
538             }
539             require(tempTotal <= 100, "total ratios must <= 100");
540             _totalRatios = tempTotal;
541         }
542 
543         proposalExecuted[_proposalId] == true;
544 
545         emit ProposalExecuted(_proposalId, pro.purpose);
546     }
```

Figure 1 Source code of *executeProposal* function (unfixed)

Recommendations	It is recommended to add verification to verify whether the proposal is passed.
Status	Fixed.



```
function executeProposal(uint256 _proposalId) public {
    require(!proposalExecuted[_proposalId], "proposal has been executed");

    Proposal storage pro = proposals[_proposalId];
    require(pro.proposer != address(0), "propsoal with this proposalId may not exist");
    require(pro.excuteTime > 0 && pro.excuteTime <= block.timestamp, "executeTime not meet");
    require(pro.sigCount >= threshold(), "proposal not meet threshold");

    if (pro.purpose == ProposalPurpose.ChangeVIP) {
        uint256 tempTotal = _totalRatios;
        for (uint256 i = 0; i < pro.vips.length; i++) {
            uint256 currRatio = _vips[pro.vips[i]];
            _vips[pro.vips[i]] = pro.ratios[i];
            if (currRatio == 0) {
                _vipAccs.push(pro.vips[i]);
            }
            // 记录历史比例
            // 影响的是第二个周期的收益比例
            // |  currMonth() | curr | next  |
            _vipHistoryRatios[pro.vips[i]][_currentMonth() + 1] = pro.ratios[i];
            _vipChangedAtMonth[pro.vips[i]][_currentMonth() + 1] = true;
            tempTotal = tempTotal - currRatio + pro.ratios[i];
            emit ChangeVIP(pro.vips[i], currRatio, pro.ratios[i]);
        }
        require(tempTotal <= 100, "total ratios must <= 100");
        _totalRatios = tempTotal;
    }

    proposalExecuted[_proposalId] = true;

    emit ProposalExecuted(_proposalId, pro.purpose);
}
}
```

Figure 2 Source code of *executeProposal* function (fixed)

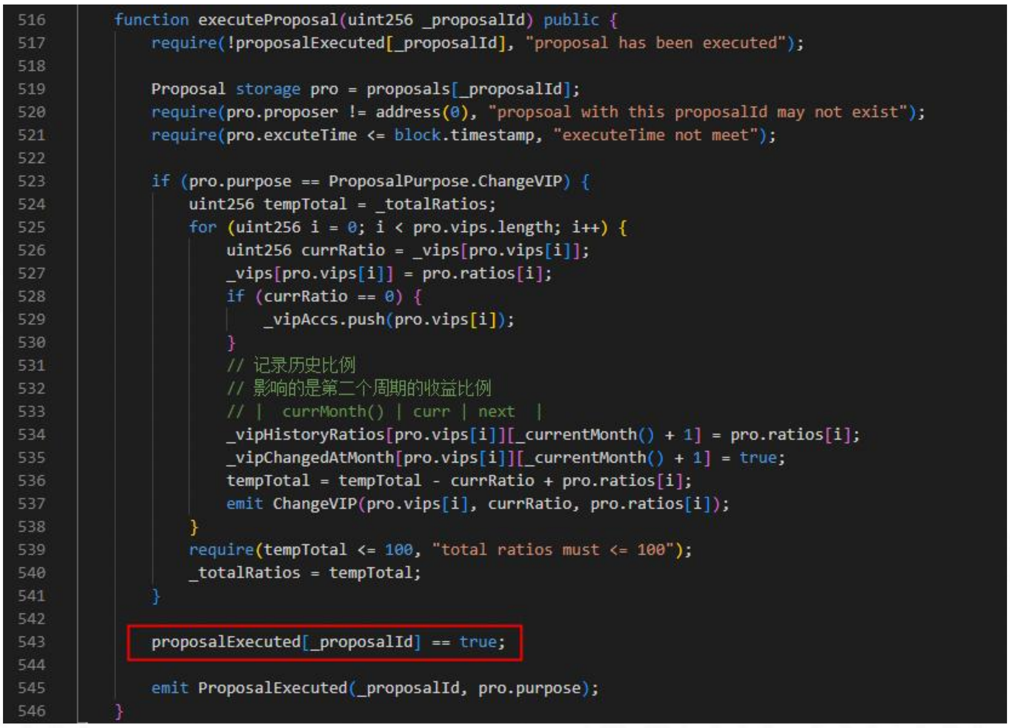
[Otter-Airdrop-2] Proposal reuse vulnerability

Severity Level Medium

Type Coding Conventions

Lines Airdrop.sol #L543

Description Incorrect use of comparison symbol as assignment symbol. Duplicate proposals can be executed. Repeated execution of the proposal can lead to the new dividend ratio being restored to the ratio of the previous proposal.

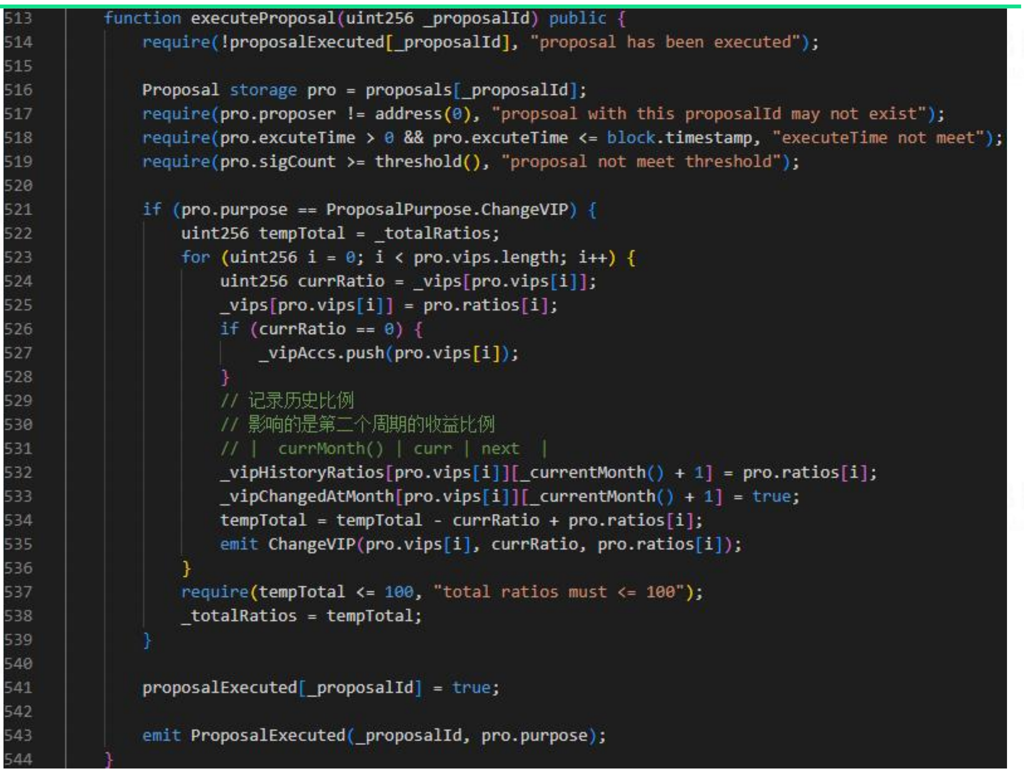


```
516     function executeProposal(uint256 _proposalId) public {
517         require(!proposalExecuted[_proposalId], "proposal has been executed");
518 
519         Proposal storage pro = proposals[_proposalId];
520         require(pro.proposer != address(0), "propsoal with this proposalId may not exist");
521         require(pro.excuteTime <= block.timestamp, "executeTime not meet");
522 
523         if (pro.purpose == ProposalPurpose.ChangeVIP) {
524             uint256 tempTotal = _totalRatios;
525             for (uint256 i = 0; i < pro.vips.length; i++) {
526                 uint256 currRatio = _vips[pro.vips[i]];
527                 _vips[pro.vips[i]] = pro.ratios[i];
528                 if (currRatio == 0) {
529                     _vipAccs.push(pro.vips[i]);
530                 }
531                 // 记录历史比例
532                 // 影响的是第二个周期的收益比例
533                 // |  currMonth() | curr | next  |
534                 _vipHistoryRatios[pro.vips[i]][_currentMonth() + 1] = pro.ratios[i];
535                 _vipChangedAtMonth[pro.vips[i]][_currentMonth() + 1] = true;
536                 tempTotal = tempTotal - currRatio + pro.ratios[i];
537                 emit ChangeVIP(pro.vips[i], currRatio, pro.ratios[i]);
538             }
539             require(tempTotal <= 100, "total ratios must <= 100");
540             _totalRatios = tempTotal;
541         }
542 
543         proposalExecuted[_proposalId] == true;
544 
545         emit ProposalExecuted(_proposalId, pro.purpose);
546     }
```

Figure 3 Source code of *executeProposal* function (unfixed)

Recommendations It is recommended to use the assignment symbol.

Status Fixed.



```
function executeProposal(uint256 _proposalId) public {
    require(!proposalExecuted[_proposalId], "proposal has been executed");

    Proposal storage pro = proposals[_proposalId];
    require(pro.proposer != address(0), "propsoal with this proposalId may not exist");
    require(pro.excuteTime > 0 && pro.excuteTime <= block.timestamp, "executeTime not meet");
    require(pro.sigCount >= threshold(), "proposal not meet threshold");

    if (pro.purpose == ProposalPurpose.ChangeVIP) {
        uint256 tempTotal = _totalRatios;
        for (uint256 i = 0; i < pro.vips.length; i++) {
            uint256 currRatio = _vips[pro.vips[i]];
            _vips[pro.vips[i]] = pro.ratios[i];
            if (currRatio == 0) {
                _vipAccs.push(pro.vips[i]);
            }
            // 记录历史比例
            // 影响的是第二个周期的收益比例
            // |  currMonth() | curr | next  |
            _vipHistoryRatios[pro.vips[i]][_currentMonth() + 1] = pro.ratios[i];
            _vipChangedAtMonth[pro.vips[i]][_currentMonth() + 1] = true;
            tempTotal = tempTotal - currRatio + pro.ratios[i];
            emit ChangeVIP(pro.vips[i], currRatio, pro.ratios[i]);
        }
        require(tempTotal <= 100, "total ratios must <= 100");
        _totalRatios = tempTotal;
    }

    proposalExecuted[_proposalId] = true;

    emit ProposalExecuted(_proposalId, pro.purpose);
}
```

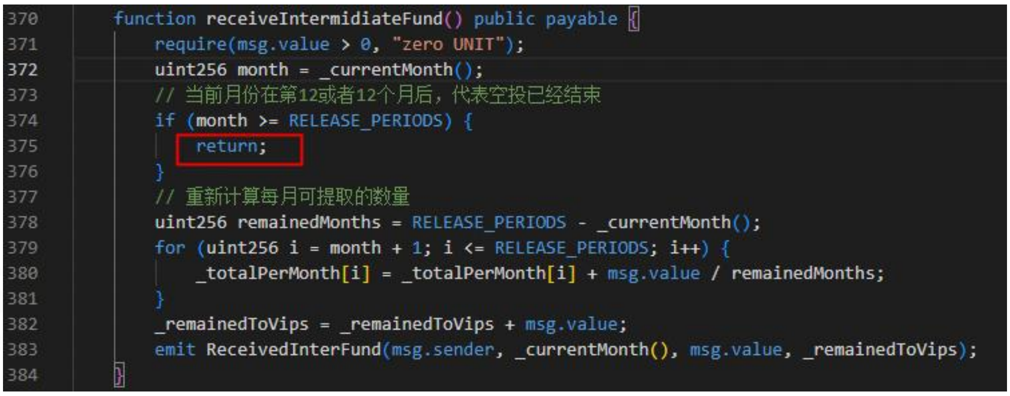
Figure 4 Source code of *executeProposal* function（fixed）

[Otter-Airdrop-3] Contract cannot withdraw funds

Severity Level	Low
Type	Business Security
Lines	Airdrop.sol #L375

Description

When the 12th month or after the 12th month, the injection of funds will not be able to withdraw. 'return' in the code is just the end function symbol, funds can still be transferred to the contract.



```
370        function receiveIntermidiateFund() public payable {
371            require(msg.value > 0, "zero UNIT");
372            uint256 month = _currentMonth();
373            // 当前月份在第12或者12个月后，代表空投已经结束
374            if (month >= RELEASE_PERIODS) {
375                return;
376            }
377            // 重新计算每月可提取的数量
378            uint256 remainedMonths = RELEASE_PERIODS - _currentMonth();
379            for (uint256 i = month + 1; i <= RELEASE_PERIODS; i++) {
380                _totalPerMonth[i] = _totalPerMonth[i] + msg.value / remainedMonths;
381            }
382            _remainedToVips = _remainedToVips + msg.value;
383            emit ReceivedInterFund(msg.sender, _currentMonth(), msg.value, _remainedToVips);
384        }
```

Figure 5 Source code of *receiveIntermidiateFund* function (unfixed)

Recommendations	It is recommended to use ‘revert()’ in the judgment function, inject funds for more than 12 months.
Status	Acknowledged.

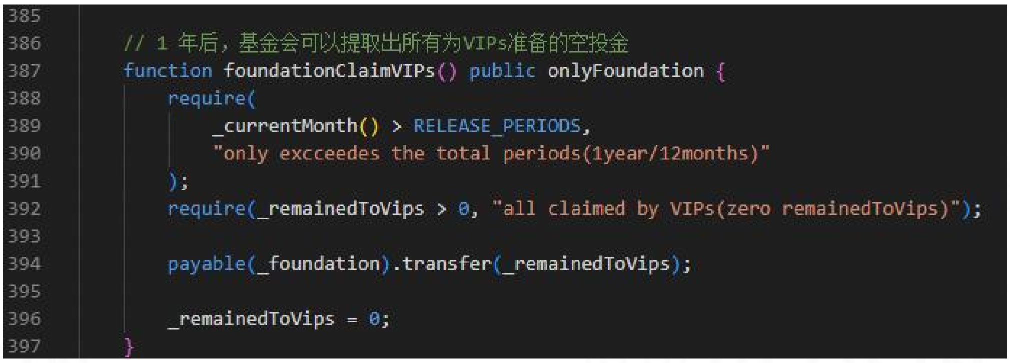
[Otter-Airdrop-4] The management role operation did not trigger an event

Severity Level Info

Type Coding Conventions

Lines Airdrop.sol #L387-397

Description Project management role operations are not logged.

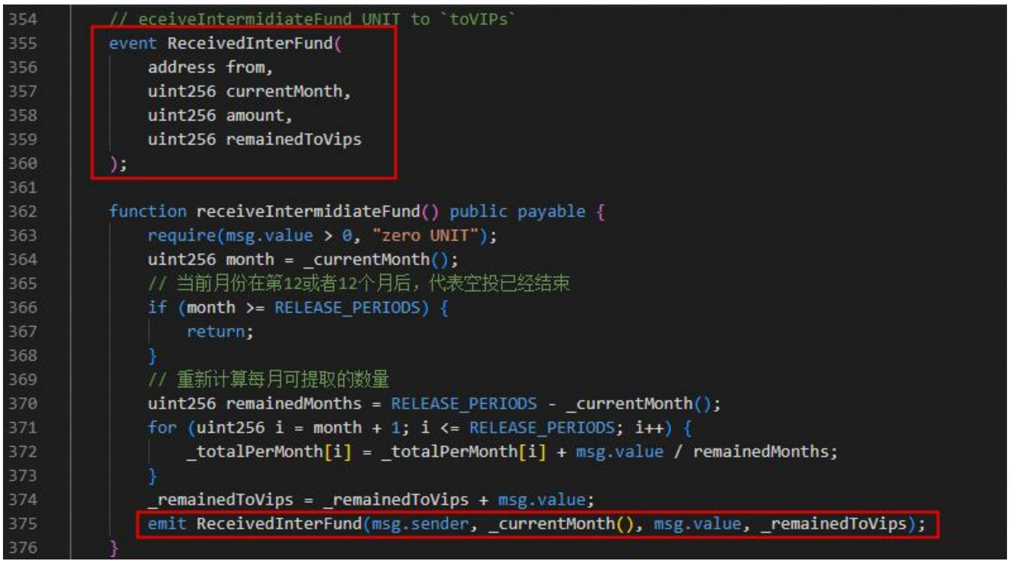


```
385
386      // 1 年后，基金会可以提取出所有为VIPs准备的空投金
387      function foundationClaimVIPs() public onlyFoundation {
388          require(
389              _currentMonth() > RELEASE_PERIODS,
390              "only excceedes the total periods(1year/12months)"
391          );
392          require(_remainedToVips > 0, "all claimed by VIPs(zero remainedToVips)");
393
394          payable(_foundation).transfer(_remainedToVips);
395
396          _remainedToVips = 0;
397      }
```

Figure 6 Source code of *receiveIntermidiateFund* function (unfixed)

Recommendations It is recommended to add corresponding events and trigger them.

Status Fixed.



```
    // eceiveIntermidiateFund UNIT to `toVIPs`
    event ReceivedInterFund(
        address from,
        uint256 currentMonth,
        uint256 amount,
        uint256 remainedToVips
    );

    function receiveIntermidiateFund() public payable {
        require(msg.value > 0, "zero UNIT");
        uint256 month = _currentMonth();
        // 当前月份在第12或者12个月后，代表空投已经结束
        if (month >= RELEASE_PERIODS) {
            return;
        }
        // 重新计算每月可提取的数量
        uint256 remainedMonths = RELEASE_PERIODS - _currentMonth();
        for (uint256 i = month + 1; i <= RELEASE_PERIODS; i++) {
            _totalPerMonth[i] = _totalPerMonth[i] + msg.value / remainedMonths;
        }
        _remainedToVips = _remainedToVips + msg.value;
        emit ReceivedInterFund(msg.sender, _currentMonth(), msg.value, _remainedToVips);
    }
```

Figure 7 Source code of *receiveIntermidiateFund* function (fixed)

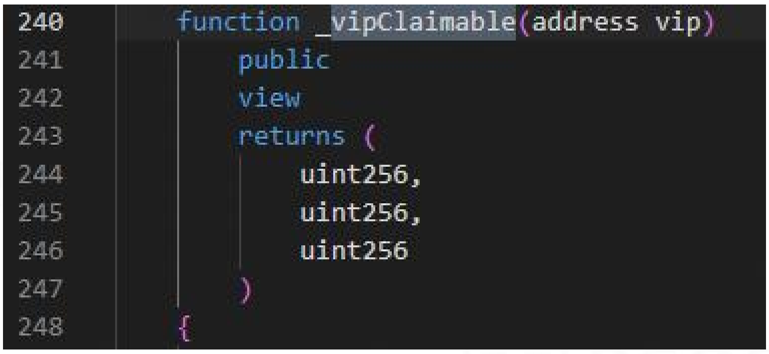
[Otter-Airdrop-5] Naming convention issue

Severity Level Info

Type Coding Conventions

Lines Airdrop.sol #L240

Description ‘*_vipClaimable*’ is not follow the naming-convention.

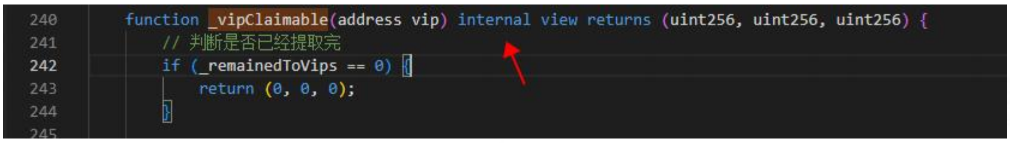


```
240    function _vipClaimable(address vip)
241        public
242        view
243        returns (
244            uint256,
245            uint256,
246            uint256
247        )
248    {
```

Figure 8 Source code of *_vipClaimable* function(unfixed)

Recommendations It is recommended to change to an internal function.

Status Fixed.



```
240    function _vipClaimable(address vip) internal view returns (uint256, uint256, uint256) {
241        // 判断是否已经提取取完
242        if (_remainedToVips == 0) {
243            return (0, 0, 0);
244        }
245
```

Figure 9 Source code of *_vipClaimable* function(fixed)

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	**Description**
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com