Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.05.04, the SlowMist security team received the Alpaca Finance team's security audit application for Alpaca Finance (Phase 1), developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Grazing Range is a magical place where Alpacas can graze on rare crops of their choice. On the Grazing Range page, users will be able to select which token rewards they wish to graze on and stake ibALPACA to receive those rewards. Grazing Range feature will make Alpaca Featured Leveraged Pool Program (FLPP) more efficient, scalable and add more value to the Alpaca ecosystem.

Initial audit files:

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6/token/GrazingRange.sol

commit: 1632dab66cb9cdb66edc886895bb996844cbea51

Final audit files:

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/main/contracts/6.12/GrazingRange.sol

commit: 1874bdadb6adc0d11f34b1d7330a9accae3c039c

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Missing event record	Others	Suggestion	Fixed
N2	Failure to follow the Checks-Effects-Interactions principle	Others	Suggestion	Fixed
N3	Unexpected reward risk	Design Logic Audit	Low	Discussed
N4	The deflationary token docking issue	Others	Suggestion	Confirmed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

GrazingRange			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
setRewardHolder	External	Can Modify State	onlyOwner
setRewardInfoLimit	External	Can Modify State	onlyOwner

GrazingRange			
addCampaignInfo	External	Can Modify State	onlyOwner
addRewardInfo	External	Can Modify State	onlyOwner
rewardInfoLen	External	-	-
campaignInfoLen	External	-	-
currentEndBlock	External	-	-
_endBlockOf	Internal	-	-
currentRewardPerBlock	External	-	-
_rewardPerBlockOf	Internal	-	-
getMultiplier	Public	-	-
pendingReward	External	-	-
_pendingReward	Internal	-	-
updateCampaign	External	Can Modify State	nonReentrant
_updateCampaign	Internal	Can Modify State	-
massUpdateCampaigns	External	Can Modify State	nonReentrant
deposit	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
_withdraw	Internal	Can Modify State	-
harvest	External	Can Modify State	nonReentrant
emergencyWithdraw	External	Can Modify State	nonReentrant
emergencyRewardWithdraw	External	Can Modify State	onlyOwner nonReentrant

4.3 Vulnerability Summary

[N1] [Suggestion] Missing event record

Category: Others

Content

In the contract, the owner can modify the rewardInfoLimit parameter through the setRewardInfoLimit function, but the event is not used for recording.

Code location:

```
function setRewardInfoLimit(uint256 _updatedRewardInfoLimit) external onlyOwner {
        rewardInfoLimit = _updatedRewardInfoLimit;
}
```

Solution

It is recommended to use events to record when modifying sensitive parameters for follow-up self-examination and community review.

Status

Fixed

[N2] [Suggestion] Failure to follow the Checks-Effects-Interactions principle

Category: Others

Content

In the event of an emergency, users can withdraw their staked assets through the emergencyWithdraw function. However, during the withdrawal process, it first transfers the staked funds to the user through the safeTransfer function, and then sets user.amount and user.rewardDebt to 0, which does not comply with the Checks-Effects-Interactions principle.

Code location:

```
function emergencyWithdraw(uint256 _campaignID) external {
    CampaignInfo storage campaign = campaignInfo[_campaignID];
    UserInfo storage user = userInfo[_campaignID][msg.sender];
    campaign.stakingToken.safeTransfer(address(msg.sender), user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
    emit EmergencyWithdraw(msg.sender, user.amount, _campaignID);
}
```

Solution

It is recommended to follow the Checks-Effects-Interactions principle and change the status first before performing the transfer operation.

Status

Fixed

[N3] [Low] Unexpected reward risk

Category: Design Logic Audit

Content

In the _updateCampaign function, if `campaign.totalStaked` is 0, the `campaign.lastRewardBlock` parameter will not be updated.

Therefore, there is a special case: if a user deposits in the reward interval of [startBlock, endBlock] without anyone taking a stake, then the `campaign.totalStaked` parameter is 0, and `campaign.lastRewardBlock` The parameters will not be updated, so `campaign.lastRewardBlock == startBlock` and accRewardPerShare is 0, then the user only needs to call the harvest function to get all the rewards from startBlock to the current block.

Code location:

```
function _updateCampaign(uint256 _campaignID) internal {
    CampaignInfo storage campaign = campaignInfo[_campaignID];
    RewardInfo[] memory rewardInfo = campaignRewardInfo[_campaignID];
    if (block.number <= campaign.lastRewardBlock) {
        return;
```

```
        }
        if (campaign.totalStaked == 0) {
            // if there is no total supply, return and use the campaign's start block
as the last reward block
            // so that ALL reward will be distributed.
            // however, if the first deposit is out of reward period, last reward
block will be its block number
            // in order to keep the multiplier = 0
            if (block.number > _endBlockOf(_campaignID, block.number)) {
                campaign.lastRewardBlock = block.number;
            }
            return;
        }
        // @dev for each reward info
        for (uint256 i = 0; i < rewardInfo.length; ++i) {
            // @dev get multiplier based on current Block and rewardInfo's end block
            // multiplier will be a range of either (current block -
campaign.lastRewardBlock)
            // or (reward info's endblock - campaign.lastRewardBlock) or 0
            uint256 multiplier = getMultiplier(campaign.lastRewardBlock,
block.number, rewardInfo[i].endBlock);
            if (multiplier == 0) continue;
            // @dev if currentBlock exceed end block, use end block as the last
reward block
            // so that for the next iteration, previous endBlock will be used as the
last reward block
            if (block.number > rewardInfo[i].endBlock) {
                campaign.lastRewardBlock = rewardInfo[i].endBlock;
            } else {
                campaign.lastRewardBlock = block.number;
            }
            campaign.accRewardPerShare =
campaign.accRewardPerShare.add(multiplier.mul(rewardInfo[i].rewardPerBlock).mul(1e12)
.div(campaign.totalStaked));
        }
    }
```

Solution

It is recommended to update campaign.lastRewardBlock when campaign.totalStaked is 0.

Status

Discussed; After feedback with the project party, the project party stated that this is the expected design.

[N4] [Suggestion] The deflationary token docking issue

Category: Others

Content

Users can transfer stakingToken into the contract through the deposit function. Under normal circumstances, the number of satkingToken transferred by the user is the same as the `_amount` parameter passed in. But if stakingToken is a deflationary token, the number of tokens transferred by the user may be different from the number of tokens actually received in the contract.

```
    function deposit(uint256 _campaignID, uint256 _amount) external nonReentrant {
        CampaignInfo storage campaign = campaignInfo[_campaignID];
        UserInfo storage user = userInfo[_campaignID][msg.sender];
        _updateCampaign(_campaignID);
        if (user.amount > 0) {
            uint256 pending =
user.amount.mul(campaign.accRewardPerShare).div(1e12).sub(user.rewardDebt);
            if (pending > 0) {
                campaign.rewardToken.safeTransfer(address(msg.sender), pending);
            }
        }
        if (_amount > 0) {
            campaign.stakingToken.safeTransferFrom(address(msg.sender),
address(this), _amount);
            user.amount = user.amount.add(_amount);
            campaign.totalStaked = campaign.totalStaked.add(_amount);
        }
        user.rewardDebt = user.amount.mul(campaign.accRewardPerShare).div(1e12);
        emit Deposit(msg.sender, _amount, _campaignID);
    }
```

Solution

It is recommended to check the balance of the contract before and after the user transfer.

Status

Confirmed; After communication with the project party, the project party indicated that the contract will not be docked with deflationary tokens.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002105060002	SlowMist Security Team	2021.05.04 - 2021.05.06	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 low risk, 3 suggestion vulnerabilities. And 1 suggestion vulnerabilities were confirmed and being fixed; 1 low risk vulnerabilities were discussed; All other findings were fixed. The code was not deployed to the mainnet.

6 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