AstraSec

Campie

Security Audit Report

July 26, 2024

Contents

1 Introduction

1.1 About Campie

Developed by `Magpie`, `Campie` is a DeFi platform developed atop Camelot DEX, which emerges as a pioneering SubDAO initiative, thoughtfully designed to bolster Camelot DEX and the Arbitrum kingdom.

1.2 Audit Scope

First Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/Campie/tree/grailRush2
- Commit ID: 118fa2e

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/magpiexyz/Campie/tree/grailRush2
- Commit ID: 7d6ba8e

Note this audit only covers the CamelotStaking.sol, mGrailConvertor.sol, and MasterCampie.sol contracts.

Second Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/Campie/pull/21/
- Commit ID: 5540946

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/magpiexyz/Campie/pull/21/
- Commit ID: 6a9fe05

Third Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/Campie/pull/40/
- Commit ID: ff4dd83

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/magpiexyz/Campie/pull/40/
- Commit ID: 3c559c1

Forth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/Campie/pull/41/
- Commit ID: a20206c

This audit only covers the code change for the `MasterCampie` contract. There is no issue observed in this audit, so no fix is committed.

1.3 Changelog

Version	Date
First Audit	April 5, 2024
Second Audit	May 20, 2024
Third Audit	June 20, 2024
Forth Audit	July 26, 2024

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Campie` project. Throughout this audit, we identified several issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	1	-	-	1
Medium	1	-	-	1
Low	1	1	-	-
Informational	1	-	-	1
Undetermined	-	-	-	-

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~H-1~~ Improper Logic of mGrailConvertor::convert()

~~M-1~~ Revisited Reward Distribution in CamelotStaking

L-1 Potential Risks Associated with Centralization

~~I-1~~ Meaningful Events for Key Operations

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[H-1] Improper Logic of mGrailConvertor::convert()

Target	Category	IMPACT	LIKELIHOOD	STATUS
mGrailConvertor.sol	Business Logic	High	Medium	Addressed

The `mGrailConvertor::convert()` function is designed to process the deposits of the `grail` token and mint the `mGrail` token simultaneously. When the input parameter `_mode` is set to `stakeMode`, it will deposit the minted `mGrail` token to the `masterCampie` contract on behalf of the user. While examining the staking logic, we notice the `mGrail` token is directly minted to the `masterCampie` contract (line 80). At the same time, we observe the `MasterCampie::depositFor()` function attempts to extract the `mGrail` token again from `msg.sender` (line 329), which will cause the transaction to revert.

mGrailConvertor::convert()

```
68 function convert(
69     address _for,
70     uint256 _amount,
71     uint256 _mode
72 ) external whenNotPaused nonReentrant {
73     if (_amount == 0) revert ZeroNotAllowed();

75     IERC20(grail).safeTransferFrom(msg.sender, address(this), _amount);
76     _convert(_amount);

78     if (_mode == stakeMode) {
79         if (masterCampie == address(0)) revert MasterCampieNotSet();
80         IMintableERC20(mGrail).mint(address(masterCampie), _amount);
81         IMasterCampie(masterCampie).depositFor(
82             mGrail,
83             _for,
84             _amount
85         );
86     } else {
87         IMintableERC20(mGrail).mint(_for, _amount);
88     }
89     emit mGrailConverted(_for, _amount, _mode);
90 }
```

MasterCampie::depositFor()

```
321 function depositFor(
322     address _stakingToken,
323     address _for,
```

```
    uint256 _amount
) external whenNotPaused nonReentrant {
    PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
    IMintableERC20(pool.receiptToken).mint(_for, _amount);

    IERC20(pool.stakingToken).safeTransferFrom(address(msg.sender), address(this
        ), _amount);
    emit Deposit(_for, _stakingToken, pool.receiptToken, _amount);
}
```

Remediation Properly improve the implementation of the convert() function.

[M-1] Revisited Reward Distribution in CamelotStaking

Target	Category	IMPACT	LIKELIHOOD	STATUS
CamelotStaking.sol	Business Logic	Medium	Medium	Addressed

By design, the `depositLp()` function accepts the deposits of the `Camelot LP` tokens and deposits them into the corresponding `Camelot NFTPool` to earn `GRAIL/xGRAIL` as rewards. It calls `_harvestBatchLp-Rewards()` (line 217) to harvest and distribute rewards. Upon thorough examination of the current implementation, we identify a vulnerability that needs to be improved. The `_harvestBatchLpRewards()` function calls `harvestPosition()` (line 245) to harvest and calculate rewards based on the `GRAIL/xGRAIL` balance changes in the `CamelotStaking` contract. However, the `addToPosition()` function is called (line 212) before to deposit the `Camelot LP` tokens into the `Camelot NFTPool`, and it calls `_harvestPosition()` (line 584) (which is equivalent to `harvestPosition()`), sending rewards to the `CamelotStaking` contract. As a result, the second call to `harvestPosition()` in `_harvestBatchLpRewards()` does not yield any reward, causing them to be locked in the CamelotStaking contract without proper distribution.

CamelotStaking::depositLp()

```
191 function depositLp(
192     address _lp,
193     address _for,
194     address _from,
195     uint256 _amount
196 ) external override nonReentrant whenNotPaused _onlyActivePoolHelper(_lp){
197     Pool storage poolInfo = pools[_lp];

199     IERC20(poolInfo.lp).safeTransferFrom(_from, address(this), _amount);

201     if(lpPostionTokenId[_lp] == 0)
202     {
203         uint256 lpNextTokenId = INFTPool(poolInfo.lpNftPool).lastTokenId() + 1;
```

```
        lpPostionTokenId[_lp] = lpNextTokenId;

        IERC20(poolInfo.lp).approve(poolInfo.lpNftPool, _amount);
        INFTPool(poolInfo.lpNftPool).createPosition(_amount,0);
    }
    else
    {
        IERC20(poolInfo.lp).approve(poolInfo.lpNftPool, _amount);
        INFTPool(poolInfo.lpNftPool).addToPosition(lpPostionTokenId[_lp],
            _amount);
    }

    address[] memory _lps = new address[](1);
    _lps[0] = _lp;
    _harvestBatchLpRewards( _lps );

    // mint the receipt to the user driectly
    IMintableERC20(poolInfo.receiptToken).mint(_for, _amount);

    emit NewLpDeposit(_for, _lp, _amount, poolInfo.receiptToken, _amount);
}

function _harvestBatchLpRewards(
    address[] memory _lps
) internal {

    if(_lps.length == 0) return;

    for(uint256 i = 0; i < _lps.length; i++)
    {
        Pool storage poolInfo = pools[_lps[i]];
        if ((poolInfo.lastHarvestTime + harvestTimeGap) > block.timestamp)
            return;

        if (!pools[_lps[i]].isActive) revert OnlyActivePool();
        poolInfo.lastHarvestTime = block.timestamp;

        uint256 _tokenId = lpPostionTokenId[_lps[i]];
        if(_tokenId == 0) revert NoOpenPositionForLp();

        uint256 xGrailBeforeBalance =  xGrailToken.balanceOf(address(this));
        uint256 grailBeforeBalance = IERC20(GRAIL).balanceOf(address(this));

        INFTPool(poolInfo.lpNftPool).harvestPosition(_tokenId);

        uint256 xGrailAfterBalance =  xGrailToken.balanceOf(address(this));
        uint256 grailAfterBalance = IERC20(GRAIL).balanceOf(address(this));

        if((grailAfterBalance - grailBeforeBalance) > 0)
        {
```

```
            IERC20(GRAIL).safeApprove(address(rewardDistributor), (
                grailAfterBalance - grailBeforeBalance));
            IRewardDistributor(rewardDistributor).sendYieldBoosterRewards(
                poolInfo.rewarder, GRAIL, (grailAfterBalance - grailBeforeBalance
                ));
        }

        if((xGrailAfterBalance - xGrailBeforeBalance) > 0)
        {
            xGrailToken.approve(address(rewardDistributor), (xGrailAfterBalance
                - xGrailBeforeBalance));
            IRewardDistributor(rewardDistributor).sendYieldBoosterRewards(
                poolInfo.rewarder, address(xGrailToken), (xGrailAfterBalance -
                xGrailBeforeBalance));
        }
    }
}
```

NFTPool::addToPosition()

```
function addToPosition(uint256 tokenId, uint256 amountToAdd) external
    nonReentrant {
    _requireOnlyOperatorOrOwnerOf(tokenId);
    require(amountToAdd > 0, "0 amount"); // addToPosition: amount cannot be
        null

    _updatePool();
    address nftOwner = ERC721.ownerOf(tokenId);
    _harvestPosition(tokenId, nftOwner);

    ...
}
```

Remediation Improve the implementation of the `depositLp()` function to properly harvest and distribute the rewards.

[L-1] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Low	Low	Acknowledged

In the `Campie` protocol, the existence of a series of privileged accounts introduces centralization risks, as they hold significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged accounts.

Example Privileged Operations in Campie

```
function setPoolManagerStatus(address _account, bool _allowedManager) external
    onlyOwner {
    PoolManagers[_account] = _allowedManager;

    emit PoolManagerStatus(_account, PoolManagers[_account]);
}

function setCampie(address _campie) external onlyOwner {
    if (address(campie) != address(0)) revert CampieSetAlready();

    if (!Address.isContract(_campie)) revert MustBeContract();

    campie = IERC20(_campie);
    emit CampieSet(_campie);
}

function setVlCampie(address _vlCampie) external onlyOwner {
    address oldvlCampie = address(vlCampie);
    vlCampie = IVLCampie(_vlCampie);
    emit VlCampieUpdated(address(vlCampie), oldvlCampie);
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been confirmed by the team.

[I-1] Meaningful Events for Key Operations

Target	Category	IMPACT	LIKELIHOOD	STATUS
MasterCampie.sol	Coding Practices	N/A	N/A	Addressed

The `event` feature is vital for capturing runtime dynamics in a contract. Upon emission, `events` store transaction arguments in logs, supplying external analytics and reporting tools with crucial information. They play a pivotal role in scenarios like modifying system-wide parameters or handling token operations.

However, in our examination of protocol dynamics, we observed that certain key operations lack meaningful events to document their changes. We highlight the representative routines below.

MasterCampie::updateWhitelistedAllocManager()

```
function updateWhitelistedAllocManager(
    address _account,
    bool _allowed
) external onlyOwner {
    AllocationManagers[_account] = _allowed;
}
```

Remediation Ensure the proper emission of meaningful events containing accurate information to promptly reflect state changes.

4 Appendix

4.1 About AstraSec

`AstraSec` is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, `AstraSec` maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. `AstraSec`'s comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI