AstraSec

Cakepie

Security Audit Report

May 4, 2024

Contents

1 Introduction

1.1 About Cakepie

`Cakepie` is an advanced SubDAO created by the `Magpie` Kitchen to enhance the long-term sustainability of `PancakeSwap`'s `veCAKE` design. The primary objective of `Cakepie` is to accumulate `CAKE` tokens and lock them as `veCAKE`, helping to decrease its circulating supply. This allows `Cakepie` to capitalize on `PancakeSwap`'s structure, optimizing governance power and offering passive income opportunities for DeFi users.

`Cakepie` provides a platform for users to deposit their assets and automatically receive optimized APR as liquidity providers. Simultaneously, it offers an efficient way for `PancakeSwap` voters to gain voting power and earn passive income via the `CKP` token.

1.2 Audit Scope

First Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/cakepie_contract/tree/fixesNTestcases
- Commit ID: 8661eae

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

- https://github.com/magpiexyz/cakepie_contract/tree/fixesNTestcases
- Commit ID: bfb0993

Second Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/cakepie_contract/pull/37

- Commit ID: 81789fc

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

- Commit ID: f78ea89

Third Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/cakepie_contract/pull/65/
- Commit ID: 737b125

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

- Commit ID: 0778d9e

Forth Audit Scope

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/cakepie_contract/pull/67/
- Commit ID: 6dfd700

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

- Commit ID: 297a489

1.3 Changelog

Version	Date
First Audit	December 30, 2023
Second Audit	January 16, 2024
Third Audit	April 8, 2024
Forth Audit	April 25, 2024

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Cakepie` 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	3	-	-	3
Medium	-	-	-	-
Low	3	1	-	2
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~~ Revisited Logic of CakepieIIFOManager::harvestIIFOFromPancake()

~~H-2~~ Revisited Logic of PancakeIIFOHelper::claim()

~~H-3~~ Revisited Logic of PancakeStakingBNBChain::depositIIFO()

~~L-1~~ Revisited Pause Functionality in Current Implementation

L-2 Potential Risks Associated with Centralization

~~L-3~~ Revisited Logic of MasterCakepie::pendingTokens()/allPendingTokens()

~~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] Revisited Logic of CakepieIIFOManager::harvestIIFOFromPancake()

Target	Category	IMPACT	LIKELIHOOD	STATUS
CakepieIIFOManager.sol	Business Logic	High	High	Addressed

In the `Cakepie IFO` protocol, the `CakepieIIFOManager` contract is designed for managing `Initial Farm Offering` (IFO) events on `PancakeSwap`, which allows the privileged owner account to generate a dedicated `PancakeIIFOHelper` for each `PancakeSwap IFO`. Users engagement with `PancakeSwap IFOs` is facilitated through interaction with the dedicated `PancakeIIFOHelper` contract, offering a streamlined and simplified user experience.

In particular, the `harvestIIFOFromPancake()` function allows the privileged owner to harvest the specified `PancakeSwap IFO`. The `PancakeStakingBNBChain::harvestIIFO()` is called (line 173) to interact with the specified `PancakeSwap IFO` through the `PancakeStakingBNBChain` contract. After further analysis, we observe the call to retrieve the pid (line 176) is improper and the correct implementation shall be `IPancakeIIFOHelper(_pancakeIIFOHelper).pid()` (line 176).

CakepieIIFOManager::harvestIIFOFromPancake()

```
163 function harvestIIFOFromPancake(address _pancakeIIFO) external onlyOwner {
164     address _pancakeIIFOHelper = iifoToHelper[_pancakeIIFO];

166     (address depositToken, address offeringToken) = IPancakeIIFOHelper(
            _pancakeIIFOHelper)
167         .getDepositnOfferingToken();

169     bool isClaimed = IPancakeIIFOHelper(_pancakeIIFOHelper).isHarvestFromPancake
            ();

171     if (isClaimed) revert AlreadyClaimed();

173     IPancakeStaking(pancakeStaking).harvestIIFO(
174         _pancakeIIFOHelper,
175         _pancakeIIFO,
176         IPancakeIIFOHelper(_pancakeIIFO).pid(),
177         depositToken,
178         offeringToken
179     );

181     IPancakeIIFOHelper(_pancakeIIFOHelper).updateStatus();

183     emit RewardClaimedFor(_pancakeIIFOHelper, _pancakeIIFO);
184 }
```

Remediation Correct the implementation of the `harvestIIFOFromPancake()` function as above mentioned.

[H-2] Revisited Logic of PancakeIIFOHelper::claim()

Target	Category	IMPACT	LIKELIHOOD	STATUS
PancakeIIFOHelper.sol	Business Logic	High	High	Addressed

As previously mentioned, each `PancakeIIFOHelper` instance is assigned to a specific `PancakeSwap IFO`. In particular, the `claim()` function is designed for users to harvest their respective rewards from the `PancakeSwap IFO` after its conclusion. While examining its logic, we identified two issues that need to be corrected.

- The validation of user input `_tokens` is improper (line 187).
- The second `_pid` parameter (line 192) of the call to `CakepieIIFOManager::releaseIIFOFromPancake()` is missing.

PancakeIIFOHelper::claim()

```
176 function claim(address[] calldata _tokens) external nonReentrant whenNotPaused {
177     (address depositToken, address offeringToken) = getDepositnOfferingToken();
178     if (!isHarvestFromPancake) revert claimPhaseNotStarted();

180     if (_tokens.length == 0) revert InvalidTokenLength();

182     for (uint8 i = 0; i < _tokens.length; i++) {
183         UserInfo storage userInfo = userInfos[_tokens[i]][msg.sender];

185         if (userInfo.userDeposit == 0) revert ZeroDeposit();

187         if (_tokens[i] != depositToken  _tokens[i] != offeringToken)
188             revert InvalidTokenAddress();

190         lastClaimVestedTime = block.timestamp;

192         ICakepieIIFOManager(cakepieIIFOManager).releaseIIFOFromPancake(address(
                pancakeIIFO));

194         _updateFor(msg.sender, _tokens[i]);

196         uint256 claimableAmt = userInfos[_tokens[i]][msg.sender].userRewards;

198         if (claimableAmt > 0) _sendReward(_tokens[i], msg.sender, claimableAmt);

200         emit UserClaimed(msg.sender, claimableAmt, offeringToken);
201     }
```

```
}
```

Remediation Correct the implementation of the `claim()` function as above mentioned.

[H-3] Revisited Logic of PancakeStakingBNBChain::depositIIFO()

Target	Category	IMPACT	LIKELIHOOD	STATUS
PancakeStakingBNBChain.sol	Business Logic	High	High	Addressed

In the `Cakepie IFO` protocol, the `PancakeStakingBNBChain` contract serves as the primary entry point for interacting with different `PancakeSwap IFOs`. While examining its logic, we observe the call to `depositIIFO()` function will always be reverted.

To provide a more in-depth explanation, we present the relevant code snippet from the contracts below. Inside the `depositIIFO()` function, the `IFOInitializableV7::depositPool()` is called (line 164) to deposit the supported asset into the `PancakeSwap IFO`. Upon further examination of the `IFOInitializableV7::depositPool()` implementation, there is an internal check is for `msg.sender` (line 249). Specifically, only users with an active `PancakeSwap Profile` are granted permission to participate in the `public sale` of the `PancakeSwap IFO`. However, the `PancakeStaking-BNBChain` contract does not have an active `PancakeSwap Profile`.

PancakeStakingBNBChain::depositIIFO()

```
153 function depositIIFO(
154     address _pancakeIIFOHelper,
155     address _pancakeIIFO,
156     uint8 _pid,
157     address _depositToken,
158     address _for,
159     uint256 _amount
160 ) external nonReentrant _onlyIIFOManager {
161     IERC20(_depositToken).safeTransferFrom(_for, address(this), _amount);
162     IERC20(_depositToken).safeIncreaseAllowance(_pancakeIIFO, _amount);

164     IIFO(_pancakeIIFO).depositPool(_amount, _pid);

166     emit DepositedIntoIFO(_pancakeIIFOHelper, _pid, _amount);
167 }
```

IFOInitializableV7::depositPool()

```
function depositPool(uint256 _amount, uint8 _pid) external override nonReentrant
     notContract {
    // Checks whether the pool id is valid
    require(_pid <= MAX_POOL_ID, "Deposit: Non valid pool id");

    if (pancakeProfileAddress != address(0) && _poolInformation[_pid].saleType
        != SaleType.BASIC) {
        // Checks whether the user has an active profile when provided profile
            SC and not basic sale
        require(
            IPancakeProfile(pancakeProfileAddress).getUserStatus(msg.sender),
            "Deposit: Must have an active profile"
        );
    }
    ...
}
```

Remediation Ensure the `PancakeStakingBNBChain` contract has an active `PancakeSwap Profile`.

[L-1] Revisited Pause Functionality in Current Implementation

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Business Logic	N/A	N/A	Addressed

The `CakepieIIFOManager` contract showcases proficient code implementation and organization through the utilization of several reference contracts. Notably, it enhances the functionality by inheriting the `PausableUpgradeable` contract to support an emergency stop mechanism that can be triggered by a privileged account.

However, upon examining `CakepieIIFOManager` contract, we notice the absence of public `pause()`/`unpause()` interfaces, indicating that the emergency stop mechanism will never be activated.

CakepieIIFOManager

```
contract CakepieIIFOManager is
    Initializable,
    OwnableUpgradeable,
    ReentrancyGuardUpgradeable,
    PausableUpgradeable
{...}
```

Remediation Properly implement public `pause()/unpause()` interfaces for `CakepieIIFOManager`, `PancakeIIFOHelper`, and `PancakeStakingBNBChain` contracts.

[L-2] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Low	Low	Acknowledged

In the `Cakepie IFO` protocol, the existence of a privileged `owner` account introduces centralization risks, as it holds 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 account.

PancakeStakingBNBChain

```
function setVeCakeRewarder(
    address _veCakeShare,
    address _veRevenueShare,
    address _gaugeVoting
) external onlyOwner {
    ...
}

function setGaugeVoting(address _gaugeVoting) external onlyOwner {
    if (_gaugeVoting == address(0)) revert AddressZero();
    gaugeVoting = IGaugeVoting(_gaugeVoting);
}

function setCakepieIIFOManager(address _cakepieIIFOManager) external onlyOwner {
    if (_cakepieIIFOManager == address(0)) revert AddressZero();
    cakepieIIFOManager = _cakepieIIFOManager;
}
```

In the `Cakepie Rewarder` implementation, the existence of a privileged `owner` account also introduces centralization risks, as it holds 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 account.

StreamRewarder

```
function setRewardQueuerStatus(address _rewardQueuer, bool status) external
    onlyOwner {
    isRewardQueuer[_rewardQueuer] = status;

    emit QueuerStatusUpdated(_rewardQueuer, status);
}

function setMasterCakepie(address _masterCakepie) external onlyOwner {
    address oldMasterCakepie = masterCakepie;
```

```
    masterCakepie = _masterCakepie;

    emit MasterCakepieUpdated(oldMasterCakepie, _masterCakepie);
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. 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 and the multi-sig mechanism will be used to mitigate it.

[L-3] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Low	Low	Acknowledged

In the `Cakepie Rewarder` implementation, the existence of a privileged `owner` account introduces centralization risks, as it holds 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 account.

StreamRewarder

```
function setRewardQueuerStatus(address _rewardQueuer, bool status) external
    onlyOwner {
    isRewardQueuer[_rewardQueuer] = status;

    emit QueuerStatusUpdated(_rewardQueuer, status);
}

function setMasterCakepie(address _masterCakepie) external onlyOwner {
    address oldMasterCakepie = masterCakepie;
    masterCakepie = _masterCakepie;

    emit MasterCakepieUpdated(oldMasterCakepie, _masterCakepie);
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. 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 and the multi-sig mechanism will be used to mitigate it.

[L-4] Revisited Logic of MasterCakepie::pendingTokens()/allPendingTokens()

Target	Category	IMPACT	LIKELIHOOD	STATUS
MasterCakepie.sol	Business Logic	Low	Low	Addressed

The `MasterCakepie` contract is designed to implement an incentive mechanism that encourages users to stake supported assets. As a result of this staking activity, participants receive rewards in the form of various tokens. Specifically, the `pendingTokens()` function is utilized by users to query the pending rewards associated with the specified `_stakingToken` and `_rewardToken`.

Upon scrutinizing its logic, we've identified that the current implementation neglects the potential existence of two distinct rewarders (i.e., the current rewarder and the legacy rewarder) in specific scenarios. In such instances, the accuracy of the returned pending rewards may be compromised.

MasterCakepie::pendingTokens()

```
function pendingTokens(
    address _stakingToken,
    address _user,
    address _rewardToken
)
    external
    view
    returns (
        uint256 pendingCakepie,
        address bonusTokenAddress,
        string memory bonusTokenSymbol,
        uint256 pendingBonusToken
    )
{
    PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
    pendingCakepie = _calCakepieReward(_stakingToken, _user);

    // If it's a multiple reward farm, we return info about the specific bonus token
    if (address(pool.rewarder) != address(0) && _rewardToken != address(0)) {
        (bonusTokenAddress, bonusTokenSymbol) = (
            _rewardToken,
            IERC20Metadata(_rewardToken).symbol()
        );
        pendingBonusToken = IBaseRewardPool(pool.rewarder).earned(_user, _rewardToken);
    }
}
```

Remediation Consider the potential existence of two distinct rewarders during the pending rewards calculation in the `pendingTokens()/allPendingTokens()` functions.

[I-1] Meaningful Events for Key Operations

Target	Category	IMPACT	LIKELIHOOD	STATUS
MasterCakepie.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 privileged routines lack meaningful events to document their changes. We highlight the representative routines below.

MasterCakepie

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

function setPancakeV3Helper(address _pancakeV3Helper) external onlyOwner {
    pancakeV3Helper = IPancakeV3Helper(_pancakeV3Helper);
}

function setLegacyRewarder(address _stakingToken, address _legacyRewarder)
    external onlyOwner {
    legacyRewarders[_stakingToken] = _legacyRewarder;
}
```

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