



Cakepie

Security Audit Report

November 18, 2024

Contents

1	Introduction	3
1.1	About Cakepie	3
1.2	Audit Scope	3
1.3	Changelog	4
2	Overall Assessment	5
3	Vulnerability Summary	6
3.1	Overview	6
3.2	Security Level Reference	7
3.3	Vulnerability Details	8
4	Appendix	17
4.1	About AstraSec	17
4.2	Disclaimer	17
4.3	Contact	17

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

	LINK	Base Commit	Final Commit
1	https://github.com/magpiexyz/cakepie.../fixesNTestcases	8661eae	bfb0993
2	https://github.com/magpiexyz/cakepie_contract/pull/37	81789fc	f78ea89
3	https://github.com/magpiexyz/cakepie_contract/pull/65	737b125	0778d9e
4	https://github.com/magpiexyz/cakepie_contract/pull/67	6dfd700	297a489
5	https://github.com/magpiexyz/cakepie_contract/pull/71	730b49a	93ffc8c
6	https://github.com/magpiexyz/cakepie_contract/pull/84	8ce2d1e	-
7	https://github.com/magpiexyz/cakepie_contract/pull/88	e60f860	51293a0
8	https://github.com/magpiexyz/cakepie_contract/pull/86	a4b9a99	40add2b
9	https://github.com/magpiexyz/cakepie_contract/pull/92	7c26883	0e104f8
10	https://github.com/magpiexyz/cakepie_contract/pull/115	4735c37	b1ea76b
11	https://github.com/magpiexyz/cakepie_contract/pull/119	b4ed567	4471dbe
12	https://github.com/magpiexyz/cakepie_contract/pull/115	22a09a7	7e496a8
13	https://github.com/magpiexyz/cakepie_contract/pull/127	9a14662	3032206

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
Fifth Audit	May 14, 2024
Sixth Audit	May 30, 2024
Seventh Audit	June 15, 2024
Eighth Audit	June 24, 2024
Ninth Audit	June 26, 2024
Tenth Audit	August 24, 2024
Eleventh Audit	September 18, 2024
Twelfth Audit	September 30, 2024
Thirteenth Audit	November 13, 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	4	1	-	3
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()
- [L-4](#) Improved Reward Send Logic in RewardDistributor::sendVeReward()
- [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(
167         _pancakeIIFOHelper)
168         .getDepositnOfferingToken();

169     bool isClaimed = IPancakeIIFOHelper(_pancakeIIFOHelper).isHarvestFromPancake
170         ();

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();
179
180     if (_tokens.length == 0) revert InvalidTokenLength();
181
182     for (uint8 i = 0; i < _tokens.length; i++) {
183         UserInfo storage userInfo = userInfos[_tokens[i]][msg.sender];
184
185         if (userInfo.userDeposit == 0) revert ZeroDeposit();
186
187         if (_tokens[i] != depositToken || _tokens[i] != offeringToken)
188             revert InvalidTokenAddress();
189
190         lastClaimVestedTime = block.timestamp;
191
192         ICakepieIIFOManager(cakepieIIFOManager).releaseIIFOFromPancake(address(
            pancakeIIFO));
193
194         _updateFor(msg.sender, _tokens[i]);
195
196         uint256 claimableAmt = userInfos[_tokens[i]][msg.sender].userRewards;
197
198         if (claimableAmt > 0) _sendReward(_tokens[i], msg.sender, claimableAmt);
199
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()

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

247     if (pancakeProfileAddress != address(0) && _poolInformation[_pid].saleType
        != SaleType.BASIC) {
248         // Checks whether the user has an active profile when provided profile
            SC and not basic sale
249         require(
250             IPancakeProfile(pancakeProfileAddress).getUserStatus(msg.sender),
251             "Deposit: Must have an active profile"
252         );
253     }
254     ...
255 }
```

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

```
13 contract CakepieIIFOManager is
14     Initializable,
15     OwnableUpgradeable,
16     ReentrancyGuardUpgradeable,
17     PausableUpgradeable
18 {...}
```

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

```
269 function setVeCakeRewarder(  
270     address _veCakeShare,  
271     address _veRevenueShare,  
272     address _gaugeVoting  
273 ) external onlyOwner {  
274     ...  
275 }  
  
277 function setGaugeVoting(address _gaugeVoting) external onlyOwner {  
278     if (_gaugeVoting == address(0)) revert AddressZero();  
279     gaugeVoting = IGaugeVoting(_gaugeVoting);  
280 }  
  
282 function setCakepieIIFOManager(address _cakepieIIFOManager) external onlyOwner {  
283     if (_cakepieIIFOManager == address(0)) revert AddressZero();  
284     cakepieIIFOManager = _cakepieIIFOManager;  
285 }
```

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

```
305 function setRewardQueuerStatus(address _rewardQueuer, bool status) external  
    onlyOwner {  
306     isRewardQueuer[_rewardQueuer] = status;  
  
308     emit QueuerStatusUpdated(_rewardQueuer, status);  
309 }  
  
311 function setMasterCakepie(address _masterCakepie) external onlyOwner {  
312     address oldMasterCakepie = masterCakepie;
```

```

313     masterCakepie = _masterCakepie;

315     emit MasterCakepieUpdated(oldMasterCakepie, _masterCakepie);
316 }

```

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] 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()

```

269 function pendingTokens(
270     address _stakingToken,
271     address _user,
272     address _rewardToken
273 )
274     external
275     view
276     returns (
277         uint256 pendingCakepie,
278         address bonusTokenAddress,
279         string memory bonusTokenSymbol,
280         uint256 pendingBonusToken
281     )
282 {
283     PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
284     pendingCakepie = _calCakepieReward(_stakingToken, _user);

```

```

286     // If it's a multiple reward farm, we return info about the specific bonus
        token
287     if (address(pool.rewarder) != address(0) && _rewardToken != address(0)) {
288         (bonusTokenAddress, bonusTokenSymbol) = (
289             _rewardToken,
290             IERC20Metadata(_rewardToken).symbol()
291         );
292         pendingBonusToken = IBaseRewardPool(pool.rewarder).earned(_user,
            _rewardToken);
293     }
294 }

```

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

[L-4] Improved Reward Send Logic in `RewardDistributor::sendVeReward()`

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

The `RewardDistributor` contract provides an external `sendVeReward()` function for the `pancakeStaking` contract to send revenue shares rewards to the rewarders. Upon examining its logic, we notice that the reward send logic can be improved to prevent potential reverts when certain conditions are met.

Below is the related code snippet. The current code implementation will convert the reward token, i.e., CAKE token to `mCake` token if the `feeInfo[i].isMCAKE` is true. However, if the `feeAmount` is 0, the execution of the `_convertToMCake()` function may revert because the `smartConvert()` function requires its input `_feeAmount` to be greater than 0 (line 148).

```

RewardDistributor::sendVeReward()

125 function sendVeReward(
126     address _rewardSource,
127     address _rewardToken,
128     uint256 _amount,
129     bool _isVeCake,
130     uint256 _minRec
131 ) external nonReentrant _onlyRewardQueuer {
132     IERC20(_rewardToken).safeTransferFrom(msg.sender, address(this), _amount);
133     uint256 _leftRewardAmount = _amount;
134     uint256 totalMCake = 0;
135     Fees[] memory feeInfo;
136
137     if (_isVeCake) feeInfo = veCakeFeeInfos;

```

```

138     else feeInfo = revenueShareFeeInfo;

140     for (uint256 i = 0; i < feeInfo.length; i++) {
141         if (feeInfo[i].isActive) {
142             address rewardToken = _rewardToken;
143             uint256 feeAmount = (_amount * feeInfo[i].value) / DENOMINATOR;
144             uint256 feeToSend = feeAmount;
145             _leftRewardAmount -= feeToSend;

147             if(feeInfo[i].isMCake) {
148                 feeToSend = _convertToMCake(feeAmount);
149                 rewardToken = mCake;
150                 totalMCake += feeToSend;
151             }

153             _distributeReward(feeInfo[i], rewardToken, feeToSend, _rewardSource,
                               true);
154         }
155     }

157     if(totalMCake < _minRec) {
158         revert minReceivedNotMet();
159     }

161     if (_leftRewardAmount > 0) {
162         IERC20(_rewardToken).safeTransfer(owner(), _leftRewardAmount);
163         emit RewardFeeDustTo(_rewardToken, owner(), _leftRewardAmount);
164     }
165 }

```

Note this issue also exists in the `sendRewards()` function of the same contract.

Remediation Execute the `_convertToMCake()` function only if the value of `feeAmount` is greater than 0.

[I-1] Meaningful Events for Key Operations

Target	Category	IMPACT	LIKELIHOOD	STATUS
MasterCakpie.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

```
899 function updateWhitelistedAllocManager(address _account, bool _allowed) external
    onlyOwner {
900     AllocationManagers[_account] = _allowed;
901 }

903 function setPancakeV3Helper(address _pancakeV3Helper) external onlyOwner {
904     pancakeV3Helper = IPancakeV3Helper(_pancakeV3Helper);
905 }

907 function setLegacyRewarder(address _stakingToken, address _legacyRewarder)
    external onlyOwner {
908     legacyRewarders[_stakingToken] = _legacyRewarder;
909 }
```

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