BLOCKSEC

Security Audit
Report for Cakepie

Date: February 20, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Magpie
Target	Cakepie

Version History

Version	Date	Description
1.0	February 20, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-02-20 10:56:23+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

This audit focuses on the Cakepie contract [1] for Magpie. Cakepie is an advanced SubDAO created by the Magpie Kitchen to enhance the long-term sustainability of PancakeSwap's ve-CAKE 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 enhanced rewards for DeFi users.

Specifically, for the version 1 and 2, only the following contracts in the repository are included in the scope of this audit. Other files are not within the scope of this audit.

- contracts/cakepie/IFO/CakepieIFOManager.sol
- contracts/cakepie/IFO/PancakeIFOHelper.sol
- contracts/cakepie/IFO/RemoteCakepieIFOManager.sol
- contracts/cakepie/IFO/RemotePancakeIFOHelper.sol
- contracts/cakepie/SmartCakeConvertor.sol
- contracts/cakepie/baseupgs/PancakeStakingBaseUpg.sol
- contracts/cakepie/briberyMarket/CakepieBribeManager.sol
- contracts/cakepie/briberyMarket/PancakeVoteManager.sol
- contracts/cakepie/core/PancakeAMLHelper.sol
- contracts/cakepie/core/PancakeStakingBNBChain.sol
- contracts/cakepie/core/PancakeStakingSideChain.sol
- contracts/cakepie/core/PancakeV3Helper.sol
- contracts/cakepie/core/VLCakepie.sol
- contracts/cakepie/core/mCakeSV.sol
- contracts/cakepie/rewards/MasterCakepie.sol
- contracts/cakepie/rewards/RewardDistributor.sol
- contracts/cakepie/rewards/StreamRewarder.sol
- contracts/cakepie/rewards/vlStreamRewarder.sol
- contracts/cakepie/tokens/CakepieReceiptToken.sol
- contracts/cakepie/crosschain/CakepieCCIPBridge.sol
- contracts/libraries/cakepie/PancakeStakingLib.sol

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA

[1] https://github.com/magpiexyz/cakepie_contract

values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Cakepie	`Version 1`	`6f68448fd04a83f0d80ef1fc9b795ce7728d2aca`
	`Version 2`	`455fe87dfc08fe31431e2886fc74a5024f01db22`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control

- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Permission management
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **eleven** potential security issues. Besides, we have **six** recommendations and **six** notes.

- High Risk: 2
- Medium Risk: 7
- Low Risk: 2
- Recommendation: 6
- Note: 6

ID	Severity	Description	Category	Status
1	Medium	Potential failures when creating `PancakeIFOHelper`	DeFi Security	Confirmed
2	Medium	Incorrect token transfer in function `_withdraw()`	DeFi Security	Fixed
3	Medium	Lack of implementation of `pause()` and `unpause()` in contract `CakepieCCIPBridge`	DeFi Security	Fixed
4	Medium	Incorrect check in function `tokenTransfer()`	DeFi Security	Fixed
5	High	Sandwich attacks when converting `Cake` to `mCake`	DeFi Security	Fixed
6	High	Potential reward loss due to evil donations	DeFi Security	Fixed
7	Low	Incorrect calculation in function `_updateVoteAndCheck()`	DeFi Security	Fixed
8	Medium	Unclaimable rewards due to forfeit when unlocking	DeFi Security	Confirmed
9	Low	Incorrect struct `Fees` in interface `IRewardDistributor`	DeFi Security	Fixed
10	Medium	Incorrect `forfeitAmount` in `mCakeSVBaseRewarder`	DeFi Security	Fixed
11	Medium	Potential reward dilutions over time	DeFi Security	Fixed
12	-	Incorrect event messages	Recommendation	Fixed
13	-	Simplify redundant calculations	Recommendation	Fixed
14	-	Fix typos	Recommendation	Fixed
15	-	Lack of invoking function `_disableInitializers()`	Recommendation	Fixed
16	-	Redundant inheritance	Recommendation	Fixed
17	-	Gas optimization by changing external calls to internal calls	Recommendation	Fixed
18	-	Sync user amounts on multiple chains	Note	-
19	-	Potential reward dilutions over time	Note	-

20	-	Status of mapping `cakeRewardToMcake`	Note	-
21	-	Harvest pools daily through automation	Note	-
22	-	The design of maximum user cap	Note	-
23	-	Potential centralization risk	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Potential failures when creating `PancakeIFOHelper`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the contract `CakepieIFOManager`, the function `createPancakeIFOHelper()` is used to create a new instance of `PancakeIFOHelper`. It will check whether `ifoToHelper[_pancakeIFO] != address(0)` to ensure that one `pancakeIFO` only has one instance of `PancakeIFOHelper`. However, the contract `IFOInitializableV8` indicates that one IFO may have different pids. In this case, this check will lead to failures when creating instances of `PancakeIFOHelper` which have the same `PancakeIFO` contract but different pools (with different pids). The contract `RemoteCakepieIFOManager` has the same problem as well.

```
174  function createPancakeIFOHelper(address _pancakeIFO, uint8 _pid) external onlyOwner {
175      if (ifoToHelper[_pancakeIFO] != address(0)) revert HelperExist();
176
177      PancakeIFOHelper newIFOHelper = new PancakeIFOHelper(
178          _pancakeIFO,
179          _pid,
180          mCakeSV,
181          pancakeStaking,
182          address(this),
183          treasuryAddress,
184          mCakeLp,
185          mCakeLpToken,
186          smartCakeConvertor
187      );
188      pancakeIFOHelpersList.push(address(newIFOHelper));
189      isIFOHelperValid[address(newIFOHelper)] = true;
190      ifoToHelper[_pancakeIFO] = address(newIFOHelper);
191      emit IFOHelperCreated(address(newIFOHelper), _pancakeIFO, _pid);
192  }
```

Listing 2.1: contracts/cakepie/IFO/CakepieIFOManager.sol

Impact This will lead to potential failures when creating `PancakeIFOHelper`.

Suggestion Add pid as a dimension in the map `ifoToHelper`.

Feedback from the project Each `pid` is corresponding to a sale. And we only allow users to participate in public sale via our contracts. Therefore, for each Pancake IFO, we'll need only one `PancakeIFOHelper` contract.

2.1.2 Incorrect token transfer in function `_withdraw()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `MasterCakepie`, the function `_withdraw()` directly transfers tokens to `msg.sender` when the flag `_isLock` is false. However, the function `_deposit()` transfers tokens from the address `_from` rather than `msg.sender` when the flag `_isLock` is false. Therefore, the function `_withdraw()` should also transfer tokens back to the address `_account`. Otherwise, tokens may be transferred to wrong addresses which finally may cause a loss to users' funds.

```
546  function _withdraw(
547      address _stakingToken,
548      address _account,
549      uint256 _amount,
550      bool _isLock
551  ) internal {
552      PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
553      UserInfo storage user = userInfo[_stakingToken][_account];
554
555      if (!_isLock && user.available < _amount) revert WithdrawAmountExceedsStaked();
556      else if (user.amount < _amount && _isLock) revert UnlockAmountExceedsLocked();
557
558      updatePool(_stakingToken);
559      _harvestCakepie(_stakingToken, _account);
560      _harvestBaseRewarder(_stakingToken, _account);
561
562      user.amount = user.amount - _amount;
563      if (!_isLock) {
564          user.available = user.available - _amount;
565          IERC20(tokenToPoolInfo[_stakingToken].stakingToken).safeTransfer(
566              address(msg.sender),
567              _amount
568          );
569      }
570      user.rewardDebt = (user.amount * pool.accCakepiePerShare) / 1e12;
571
572      pool.totalStaked -= _amount;
573
574      emit Withdraw(_account, _stakingToken, pool.receiptToken, _amount);
575  }
```

Listing 2.2: contracts/cakepie/rewards/MasterCakepie.sol

Impact Tokens may be transferred to wrong addresses which finally may cause a loss to users' funds.

Suggestion Change the `address(msg.sender)` to `address(_account)`.

2.1.3 Lack of implementation of `pause()` and `unpause()` in contract `CakepieCCIPBridge`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `CakepieCCIPBridge` inherits from the `PausableUpgradeable` contract, however, it does not implement the functions `pause()` and `unpause()`. This will lead to the result that the mechanism of pausing and unpausing can not function as expected.

```
14contract CakepieCCIPBridge is
15   Initializable,
16   OwnableUpgradeable,
17   ReentrancyGuardUpgradeable,
18   PausableUpgradeable
```

Listing 2.3: contracts/cakepie/crosschain/CakepieCCIPBridge.sol

Impact The mechanism of pausing and unpausing can not function as expected.

Suggestion Implement the functions of `pause()` and `unpause()`.

2.1.4 Incorrect check in function `tokenTransfer()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `CakepieCCIPBridge`, the function `tokenTransfer()` will refund excess native tokens to users. However, the check `0 > msg.value - fee` is incorrect, which should be `msg.value - fee > 0`. The incorrect check will cause the refund to fail.

```
110   function tokenTransfer(
111      uint64 destinationChainSelector,
112      address _receiver,
113      uint256 _amount
114   ) external payable nonReentrant whenNotPaused onlyWhitelistedChain(destinationChainSelector) {
115      if (_receiver == address(0)) revert InvalidAddress();
116
117      if (_amount == 0 || msg.value == 0) revert InvalidAmount();
118
119      IERC20(cakepie).safeTransferFrom(msg.sender, address(this), _amount);
120      IERC20(cakepie).safeIncreaseAllowance(chainlinkRouter, _amount);
121
122      (Client.EVM2AnyMessage memory evm2AnyMessage, uint256 fee) = _estimateGasFee(
123         destinationChainSelector,
124         _receiver,
125         cakepie,
126         _amount,
```

```
127             address(0)
128         );
129
130         if (fee > msg.value) revert NotEnoughBalance(msg.value, fee);
131
132         if (0 > msg.value - fee) {
133             // Calculate excess funds
134             uint256 excessFunds = msg.value - fee;
135             // Refund excess funds to the sender
136             payable(msg.sender).transfer(excessFunds);
137         }
138
139         bytes32 messageId;
140
141         messageId = IRouterClient(chainlinkRouter).ccipSend{ value: fee }(
142             destinationChainSelector,
143             evm2AnyMessage
144         );
145
146         emit TokensTransferred(
147             messageId,
148             destinationChainSelector,
149             _receiver,
150             cakepie,
151             _amount,
152             address(0),
153             fee
154         );
155     }
```

Listing 2.4: contracts/cakepie/crosschain/CakepieCCIPBridge.sol

Impact The incorrect check will cause the refund to fail.

Suggestion Change the check to `msg.value - fee > 0` instead.

2.1.5 Sandwich attacks when converting `Cake` to `mCake`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `MasterCakepie`, the functions `multiclaimFor()` and `multiclaimMCake()` allow anyone to claim rewards for others. The process of claiming rewards will convert `Cake` to `mCake` when there exist rewards of `Cake`. Furthermore, the swap will be conducted in the contract `PancakeSwap` and the slippage protection parameter `_minRecMCake` is either `0` or a parameter which can be assigned by the caller. Thus a malicious user can conduct a sandwich attack.

1. The attacker buys `mCake` with `Cake` in `PancakeSwap`.
2. The attacker claims `Cake` rewards for others with `_minRecMCake` to be `0`.

3. The process of claiming `Cake` rewards will buy `mCake` with `Cake` in `PancakeSwap`, which finally increases the price of `mCake`.

4. The attacker sells `mCake` at a higher price and gets more `Cake`.

```
435  function multiclaimFor(
436      address[] calldata _stakingTokens,
437      address[][] memory _rewardTokens,
438      address _account
439  ) external whenNotPaused {
440      uint256[] memory noTokenid = new uint256[](0);
441      _multiClaim(_stakingTokens, _account, _account, _rewardTokens, noTokenid, true, 0);
442  }
443
444  /// @notice Claims for V2 pools only whose CAKE rewards needs to be converted to MCAKE
445  function multiclaimMCake(
446      address[] calldata _stakingTokens,
447      address[][] memory _rewardTokens,
448      address _account,
449      uint256 _minRecMCake
450  ) external whenNotPaused {
451      uint256[] memory noTokenid = new uint256[](0);
452      _multiClaim(_stakingTokens, _account, _account, _rewardTokens, noTokenid, true, _minRecMCake
             );
453  }
```

Listing 2.5: contracts/cakepie/rewards/MasterCakepie.sol

```
335  function _sendReward(
336      address _cakeToken,
337      address _account,
338      address _receiver,
339      uint256 _amount,
340      uint256 _minRecMCake
341  ) internal {
342      userInfos[_cakeToken][_account].userRewards = 0;
343      uint256 mCakeReward = _convertToMCake(_amount);
344      if (mCakeReward < _minRecMCake)
345          revert minReceivedNotMet();
346
347      IERC20(mCakeToken).safeTransfer(_receiver, mCakeReward);
348      emit RewardPaid(_account, _receiver, mCakeReward, mCakeToken);
349  }
```

Listing 2.6: contracts/cakepie/rewards/CakeMCakeRewarder.sol

Impact A malicious user can conduct a sandwich attack to steal part of `Cake` rewards from others.

Suggestion Revise the logic accordingly.

2.1.6 Potential reward loss due to evil donations

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `StreamRewarder`, a user can invoke the function `donateRewards()` to donate rewards. However, since there is a precision loss in calculating the `rewardRate`, a malicious user can donate `0 wei` amount to trigger the calculation of the `rewardRate`. Donations can be made when the value of remaining is quite close to the value of duration. For example, when `duration = 101`, `remaining = 100`, old `rewardRate = 19` and the donation reward is `0 wei`, the new `rewardRate` will be `18` due to precision loss. If the evil donations are made frequently enough, the `rewardRate` may be quite small and leave part of rewards that can not be claimed. The contract `vlStreamRewarder` also has the same problem.

```
216  function donateRewards(address _rewardToken, uint256 _rewards) external nonReentrant {
217      if(!isRewardToken[_rewardToken])
218          revert InvalidToken();
219
220      IERC20(_rewardToken).safeTransferFrom(msg.sender, address(this), _rewards);
221      _provisionReward(_rewards, _rewardToken);
222      emit RewardQueued(_rewardToken, _rewards);
223
224  }
```

Listing 2.7: contracts/cakepie/rewards/StreamRewarder.sol

```
243  function _provisionReward(uint256 _rewards, address _rewardToken) internal {
244
245      _rewards = _rewards * DENOMINATOR; // to support small deciaml rewards
246
247      Reward storage rewardInfo = rewards[_rewardToken];
248
249      if (totalStaked() == 0) {
250          rewardInfo.queuedRewards = rewardInfo.queuedRewards + _rewards;
251          return ;
252      }
253
254      rewardInfo.rewardPerTokenStored = rewardPerToken(_rewardToken);
255      _rewards = _rewards + rewardInfo.queuedRewards;
256      rewardInfo.queuedRewards = 0;
257
258      if (block.timestamp >= rewardInfo.periodFinish) {
259          rewardInfo.rewardRate = _rewards / duration;
260      } else {
261          uint256 remaining = rewardInfo.periodFinish - block.timestamp;
262          uint256 leftover = remaining * rewardInfo.rewardRate;
263          _rewards = _rewards + leftover;
264          rewardInfo.rewardRate = _rewards / duration;
265      }
266      rewardInfo.lastUpdateTime = block.timestamp;
267      rewardInfo.periodFinish = block.timestamp + duration;
268
269  }
```

Listing 2.8: contracts/cakepie/rewards/StreamRewarder.sol

Impact This may leave part of rewards that can not be claimed.

Suggestion Revise the logic accordingly.

2.1.7 Incorrect calculation in function `_updateVoteAndCheck()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `PancakeVoteManager`, the function `_updateVoteAndCheck()` will compare `block.timestamp` with the result of `getCurrentPeriodEndTime()` to determine the value of `targetTime`. However, the result of `getCurrentPeriodEndTime()` will always be larger than the `block.timestamp`. Thus, the value of `targetTime` will always be `_getNextTime()`. Meanwhile, the `targetTime` is not used in the function `_updateVoteAndCheck()` except for the event `Voted()`.

```
260  function _updateVoteAndCheck(address _user, UserVote[] memory _userVotes) internal {
261      uint256 targetTime;
262      // if the current time is greater than the end time, voting will continue into the next
             period
263      if (block.timestamp >= getCurrentPeriodEndTime()) targetTime = _getNextTime() + TWOWEEK;
264      else targetTime = _getNextTime();
265
266      uint256 length = _userVotes.length;
267      int256 totalUserVote;
268
269      for (uint256 i; i < length; i++) {
270          Pool storage pool = poolInfo[_userVotes[i].pool];
271
272          int256 weight = _userVotes[i].weight;
273          totalUserVote += weight;
274
275          if (weight != 0) {
276              if (weight > 0) {
277                  if (!pool.isActive) revert PoolNotActive(); // do the check here let users can
                         still unvote their votes
278                  uint256 absVal = uint256(weight);
279                  pool.totalVoteInVlCakepie += absVal;
280                  userVotedForPoolInVlCakepie[_user][pool.pool] += absVal;
281              } else {
282                  uint256 absVal = uint256(-weight);
283                  // check there is enough voting can be unvoted
284                  if (absVal > userVotedForPoolInVlCakepie[_user][pool.pool])
285                      revert NotEnoughVote();
286                  pool.totalVoteInVlCakepie -= absVal;
287                  userVotedForPoolInVlCakepie[_user][pool.pool] -= absVal;
288              }
289          }
290
291          emit Voted(targetTime, _user, pool.pool, weight);
292      }
293
```

```
294     // update user's total vote and all vlCkp vote
295     if (totalUserVote > 0) {
296         userTotalVotedInVlCakepie[_user] += uint256(totalUserVote);
297         totalVlCakepieInVote += uint256(totalUserVote);
298     } else {
299         userTotalVotedInVlCakepie[_user] -= uint256(-totalUserVote);
300         totalVlCakepieInVote -= uint256(-totalUserVote);
301     }
302 }
```

Listing 2.9: contracts/cakepie/briberyMarket/PancakeVoteManager.sol

```
154 function getCurrentPeriodEndTime() public view returns (uint256 endTime) {
155     uint256 nextTime = _getNextTime();
156     if (block.timestamp >= nextTime - 122400) {
157         endTime = nextTime + TWOWEEK - 122400; // if the current time has passed this period's
                end time, goto next period
158     } else {
159         endTime = nextTime - 122400; // before 1 day and 10 hours of PancakeSwapEndTime (UTC +8
                22:00)
160     }
161 }
```

Listing 2.10: contracts/cakepie/briberyMarket/PancakeVoteManager.sol

Impact The calculation is incorrect and leads to abnormal functionality.

Suggestion Revise the logic accordingly.

2.1.8 Unclaimable rewards due to forfeit when unlocking

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the contract `VLCakepie`, the functions `unlock()` and `forceUnLock()` are used to unlock a finished slot for users. The unlocking process will invoke the function `_claimFromMaster-()` first to claim rewards and then invoke the function `_unlock()` to subtract the `_unlockedAmount` from the `totalAmount`, which actually represents the `totalSupply()` of `VLCakepie`.

During the process of claiming rewards, it will eventually invoke the function `_sendReward()` in the contract `VLCakepieBaseRewarder`. If the `forfeitAmount` > 0, the `forfeitAmount` will be queued as new rewards and the new rewards will be distributed over the `totalStaked()`, which is also the `totalSupply()` of `VLCakepie`. However, this distribution is prior to the decrease of the `totalSupply()` and part of the rewards cannot be claimed. This is because the `totalStaked()` contains the `_unlockedAmount`, which will not be counted in the next claiming rewards process. There exists the same problem for the contracts `mCakeSV` and `mCakeSVBaseRewarder`. Though the contracts `VLCakepieBaseRewarder` and `mCakeSVBaseRewarder` are out of scope for this audit, we notify the potential problem of these related contracts for the completeness of this audit.

```
356  function unlock(
357      uint256 _slotIndex
358  ) external override whenNotPaused nonReentrant {
359      _checkIdexInBoundary(msg.sender, _slotIndex);
360      UserUnlocking storage slot = userUnlockings[msg.sender][_slotIndex];
361
362      if (slot.endTime > block.timestamp) revert StillInCoolDown();
363
364      if (slot.amountInCoolDown == 0) revert UnlockedAlready();
365
366      _claimFromMaster(msg.sender);
367
368      uint256 unlockedAmount = slot.amountInCoolDown;
369      _unlock(unlockedAmount);
370
371      slot.amountInCoolDown = 0;
372      IERC20(cakepie).safeTransfer(msg.sender, unlockedAmount);
373
374      emit Unlock(msg.sender, block.timestamp, unlockedAmount);
375  }
```

Listing 2.11: contracts/cakepie/core/VLCakepie.sol

```
504  function _unlock(uint256 _unlockedAmount) internal {
505      IMasterCakepie(masterCakepie).withdrawVlCakepieFor(_unlockedAmount, msg.sender); // trigers
         update pool share, so happens before total amount reducing
506      totalAmountInCoolDown -= _unlockedAmount;
507      totalAmount -= _unlockedAmount;
508  }
```

Listing 2.12: contracts/cakepie/core/VLCakepie.sol

```
125  function totalSupply() public view override returns (uint256) {
126      return totalAmount;
127  }
```

Listing 2.13: contracts/cakepie/core/VLCakepie.sol

```
324  function _queueNewRewardsWithoutTransfer(uint256 _amountReward, address _rewardToken) internal
325  {
326      Reward storage rewardInfo = rewards[_rewardToken];
327      if (totalStaked() == 0) {
328          rewardInfo.queuedRewards += _amountReward;
329      } else {
330          if (rewardInfo.queuedRewards > 0) {
331              _amountReward += rewardInfo.queuedRewards;
332              rewardInfo.queuedRewards = 0;
333          }
334          rewardInfo.rewardPerTokenStored =
335              rewardInfo.rewardPerTokenStored +
336              (_amountReward * 10**vlCakepieDecimal) / totalStaked();
337      }
338      emit ForfeitRewardAdded(_amountReward, _rewardToken);
```

```
339   }
```

Listing 2.14: contracts/cakepie/rewards/VLCakepieBaseRewarder.sol

Impact It will cause part of the forfeit rewards to be left unclaimable.

Suggestion Revise the logic accordingly. A potential way to fix this is to accumulate the reward from forfeit to the param rewardInfo. queuedRewards when the reward comes from the forfeit and distribute it over the totalStaked in the next time when the reward does not come from the forfeit.

Feedback from the project The forfeited reward mechanism has been removed in the new `vlStreamRewarder`. We'll deploy this updated rewarder for locked pools, where all new rewards will queue and distribute without the forfeit mechanism. Since `mCakeSVBaseRewarder`, `VLCakepieBaseRewarder` and existing `vlStreamRewarder` are deprecated, the issue can be ignored. However, these rewarders will remain in use because they might contain unclaimed user rewards.

2.1.9 Incorrect struct `Fees` in interface `IRewardDistributor`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the interface `IRewardDistributor`, the struct `Fees` is different from the struct `Fees` in the contract `RewardDistributor`. However, the struct `Fees` is used in the function `pancakeFeeInfos()` as a return parameter. Thus, it will cause incorrect return values when other contracts invoke the function `pancakeFeeInfos()` through the interface `IRewardDistributor`.

```
31  struct Fees {
32      uint256 value; // allocation denominated by DENOMINATOR
33      address to;
34      bool isMCAKE;
35      bool isAddress;
36      bool isActive;
37  }
```

Listing 2.15: contracts/cakepie/rewards/RewardDistributor.sol

```
 8  struct Fees {
 9      uint256 value; // allocation denominated by DENOMINATOR
10      address to;
11      bool isAddress;
12      bool isActive;
13  }
```

Listing 2.16: contracts/interfaces/cakepie/IRewardDistributor.sol

```
15  function pancakeFeeInfos(uint256 index) external view returns (Fees memory);
```

Listing 2.17: contracts/interfaces/cakepie/IRewardDistributor.sol

Impact It will cause incorrect return values when other contracts invoke the function `pancakeFeeInfos()` of the contract `RewardDistributor` through the interface `IRewardDistributor`.

Suggestion Add `isMCAKE` in the struct `Fees` in the interface `IRewardDistributor`.

2.1.10 Incorrect `forfeitAmount` in `mCakeSVBaseRewarder`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `mCakeSVBaseRewarder`, the function `_calExpireForfeit()` calculates the `forfeitAmount` that should be subtracted from the users' rewards when part of users' amount is fully unlocked. However, the calculation result of the `forfeitAmount` will always be zero. This is because the `rewardableAmount` is assigned as `_amount,` which leads the result of `_amount - rewardableAmount` to be zero. As a result, the mechanism of forfeit can not function as expected.

```
367  function _calExpireForfeit(address _account, uint256 _amount) internal view returns (uint256) {
368      uint256 rewardableAmount = _amount;
369      if (rewardableAmount > _amount) revert InvalidRewardableAmount();
370
371      uint256 forfeitAmount = _amount - rewardableAmount;
372
373      if (forfeitAmount < (_amount / 1000)) {
374          // if forfeitAmount is smaller than 0.1% ignore to save gas fee
375          forfeitAmount = 0;
376          rewardableAmount = _amount;
377      }
378
379      return forfeitAmount;
380  }
```

Listing 2.18: contracts/cakepie/rewards/mCakeSVBaseRewarder.sol

Impact The reward forfeit mechanism for `mCakeSV` can not function as expected.

Suggestion Correct the calculation of the `forfeitAmount` as the contract `vlStreamRewarder` does.

2.1.11 Potential reward dilutions over time

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `StreamRewarder`, the rewards are distributed over the time using `duration` and `rewardRate` for calculation. However, when the new rewards queued are less than the old `rewardRate * duration`, the old rewards will be diluted over the time. For example, at the time of `T1`, when `duration == 100` and old `rewardRate == 20`, the total rewards of `2000` will be accumulated to `rewardPerTokenStored` at `T1 + 100`. However, if `10` rewards queued

when `remaining == 50`, the old `2000` of rewards will be diluted over the time to `T1 + 150`. The contract `vlStreamRewarder` also has the same problem. The original description of the reward dilution issue is also affected by the reward forfeit mechanism. Specifically, in the contract `vlStreamRewarder`, when `forfeitAmount` > 0, the forfeit rewards will also be queued as new rewards. These rewards are queued and distributed like regular rewards without transfers. Different from the previously described problem which can be controlled only by admin operations, the potential reward dilution problem is more affected by the reward forfeit mechanism.

```
243   function _provisionReward(uint256 _rewards, address _rewardToken) internal {
244
245       _rewards = _rewards * DENOMINATOR; // to support small deciaml rewards
246
247       Reward storage rewardInfo = rewards[_rewardToken];
248
249       if (totalStaked() == 0) {
250           rewardInfo.queuedRewards = rewardInfo.queuedRewards + _rewards;
251           return ;
252       }
253
254       rewardInfo.rewardPerTokenStored = rewardPerToken(_rewardToken);
255       _rewards = _rewards + rewardInfo.queuedRewards;
256       rewardInfo.queuedRewards = 0;
257
258       if (block.timestamp >= rewardInfo.periodFinish) {
259           rewardInfo.rewardRate = _rewards / duration;
260       } else {
261           uint256 remaining = rewardInfo.periodFinish - block.timestamp;
262           uint256 leftover = remaining * rewardInfo.rewardRate;
263           _rewards = _rewards + leftover;
264           rewardInfo.rewardRate = _rewards / duration;
265       }
266       rewardInfo.lastUpdateTime = block.timestamp;
267       rewardInfo.periodFinish = block.timestamp + duration;
268
269   }
270
271();
272   }
```

Listing 2.19: contracts/cakepie/rewards/StreamRewarder.sol

Impact The reward forfeit mechanism for `vlStreamRewarder` can result in reward dilution problems.

Suggestion Refactor the reward forfeit mechanism for `vlStreamRewarder.`

2.2 Recommendations

2.2.1 Incorrect event messages

Status Fixed in `Version 2`

Introduced by Version 1

Description In the contract MasterCakepie, the name of event DepositNotAvailable() is inaccurate since the deposit action succeeded in the function _deposit() when _isLock == true. However, "NotAvailable" represents the action failed. Additionally, in the function _withdraw(), the event Withdraw() contains a wrong _receiptToken member when _isLock == false. This is because when _isLock == false, there is no pool.receiptToken.

```
518  function _deposit(
519      address _stakingToken,
520      address _from,
521      address _for,
522      uint256 _amount,
523      bool _isLock
524  ) internal {
525      PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
526      UserInfo storage user = userInfo[_stakingToken][_for];
527
528      updatePool(_stakingToken);
529      _harvestRewards(_stakingToken, _for);
530
531      user.amount = user.amount + _amount;
532      if (!_isLock) {
533          user.available = user.available + _amount;
534          IERC20(pool.stakingToken).safeTransferFrom(address(_from), address(this), _amount);
535      }
536      user.rewardDebt = (user.amount * pool.accCakepiePerShare) / 1e12;
537
538      if (_amount > 0) {
539          pool.totalStaked += _amount;
540          if (!_isLock) emit Deposit(_for, _stakingToken, pool.receiptToken, _amount);
541          else emit DepositNotAvailable(_for, _stakingToken, _amount);
542      }
543  }
```

Listing 2.20: contracts/cakepie/rewards/MasterCakepie.sol

```
546  function _withdraw(
547      address _stakingToken,
548      address _account,
549      uint256 _amount,
550      bool _isLock
551  ) internal {
552      PoolInfo storage pool = tokenToPoolInfo[_stakingToken];
553      UserInfo storage user = userInfo[_stakingToken][_account];
554
555      if (!_isLock && user.available < _amount) revert WithdrawAmountExceedsStaked();
556      else if (user.amount < _amount && _isLock) revert UnlockAmountExceedsLocked();
557
558      updatePool(_stakingToken);
559      _harvestCakepie(_stakingToken, _account);
560      _harvestBaseRewarder(_stakingToken, _account);
561
```

```
562     user.amount = user.amount - _amount;
563     if (!_isLock) {
564         user.available = user.available - _amount;
565         IERC20(tokenToPoolInfo[_stakingToken].stakingToken).safeTransfer(
566             address(msg.sender),
567             _amount
568         );
569     }
570     user.rewardDebt = (user.amount * pool.accCakepiePerShare) / 1e12;
571
572     pool.totalStaked -= _amount;
573
574     emit Withdraw(_account, _stakingToken, pool.receiptToken, _amount);
575 }
```

Listing 2.21: contracts/cakepie/rewards/MasterCakepie.sol

Suggestion Revise the logic.

2.2.2 Simplify redundant calculations

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contracts `mCakeSV` and `VLCakepie`, the calculation in function `balanceOf()` is `getUserTotalLocked(_user) + getUserAmountInCoolDown(_user)`, which is redundant. Itcan be simplified to be `masterCakepie.stakingInfo(address(this), _user)`. Meanwhile, in the function `expectedPenaltyAmountByAccount()`, there is no need to subtract the `slot.startTime` when comparing the value of `(block.timestamp - slot.startTime)` with `(slot.endTime - slot.startTime)`.

```
107 function balanceOf(address _user) public view override returns (uint256) {
108     return getUserTotalLocked(_user) + getUserAmountInCoolDown(_user);
109 }
```

Listing 2.22: contracts/cakepie/core/mCakeSV.sol

```
119 function getUserTotalLocked(address _user) public view override returns (uint256 _lockAmount) {
120     // needs fixing
121     (uint256 _amountInMasterCakepie, ) = IMasterCakepie(masterCakepie).stakingInfo(
122         address(this),
123         _user
124     );
125     _lockAmount = _amountInMasterCakepie - getUserAmountInCoolDown(_user);
126 }
```

Listing 2.23: contracts/cakepie/core/mCakeSV.sol

```
264 function expectedPenaltyAmountByAccount(address account, uint256 _slotIndex) public view
        returns(uint256 penaltyAmount, uint256 amountToUser) {
265     UserUnlocking storage slot = userUnlockings[account][_slotIndex];
266
```

```
267     uint256 coolDownAmount = slot.amountInCoolDown;
268     uint256 baseAmountToUser = slot.amountInCoolDown / 5;
269     uint256 waitingAmount = coolDownAmount - baseAmountToUser;
270
271     uint256 unlockFactor = 1e12;
272     if (
273         (block.timestamp - slot.startTime) <=
274         (slot.endTime - slot.startTime)
275     )
276         unlockFactor =
277             (((block.timestamp - slot.startTime) * 1e12) /
278                 (slot.endTime - slot.startTime)) **
279                 2 /
280             1e12;
281
282     uint256 unlockAmount = (waitingAmount * unlockFactor) / 1e12;
283     amountToUser = baseAmountToUser + unlockAmount;
284     penaltyAmount = coolDownAmount - amountToUser;
285 }
```

Listing 2.24: contracts/cakepie/core/VLCakepie.sol

Suggestion Refactor redundant calculations.

2.2.3 Fix typos

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `StreamRewarder`, the public parameter `receipTokenDecimal` should be `receiptTokenDecimal` instead. And in the contract `VLCakepie`, the name of the function `setMasterChief()` should be `setMasterChef()`. Meanwhile in the contract `RewardDistributor`, the error `OnlyRewardQeuer()` should be `OnlyRewardQueuer()`, and the `_onlyRewardQeuer()` should be `_onlyRewardQueuer()` instead.

```
36  uint256 public receipTokenDecimal;
```

Listing 2.25: contracts/cakepie/rewards/StreamRewarder.sol

```
445 function setMasterChief(address _masterCakepie) external onlyOwner {
446     if (_masterCakepie == address(0)) revert InvalidAddress();
447     address oldChief = masterCakepie;
448     masterCakepie = _masterCakepie;
449
450     emit NewMasterChiefUpdated(oldChief, masterCakepie);
451 }
```

Listing 2.26: contracts/cakepie/core/VLCakepie.sol

```
70   error OnlyRewardQeuer();
```

Listing 2.27: contracts/cakepie/rewards/RewardDistributor.sol

```
107   modifier _onlyRewardQeuer() {
108       if (msg.sender != pancakeStaking) revert OnlyRewardQeuer();
109       _;
110   }
```

Listing 2.28: contracts/cakepie/rewards/RewardDistributor.sol

Suggestion Correct the typos.

2.2.4 Lack of invoking function `_disableInitializers()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `_disableInitializers()` is not called in the constructor of contract `RewardDistributor`. In this case, anyone can invoke the function `initialize()` of the implementation contract after it has been deployed, which may bring risks in the future.

Suggestion Invoke the function `_disableInitializers()` in the constructor.

2.2.5 Redundant inheritance

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `RewardDistributor` inherits from the contract `PausableUpgradeable`. However, the contract `RewardDistributor` does not use the functions `pause()` and `unpause()` nor use the modifier `whenNotPaused`. Thus, the inheritance is redundant.

```
21contract RewardDistributor is
22   Initializable,
23   OwnableUpgradeable,
24   ReentrancyGuardUpgradeable,
25   PausableUpgradeable
```

Listing 2.29: contracts/cakepie/rewards/RewardDistributor.sol

Suggestion Remove the redundant inheritance.

2.2.6 Gas optimization by changing external calls to internal calls

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contracts `VLCakepie` and `mCakeSV`, the function `totalLocked()` externally calls the function `totalSupply()`. However, the function `totalLocked()` is public and can be called through internal calls. Changing external calls to internal calls can save extraneous gas usages.

```
134   function totalLocked() public view override returns (uint256) {
135       return this.totalSupply() - this.totalAmountInCoolDown();
136   }
```

Listing 2.30: contracts/cakepie/core/VLCakepie.sol

```
125   function totalSupply() public view override returns (uint256) {
126       return totalAmount;
127   }
```

Listing 2.31: contracts/cakepie/core/VLCakepie.sol

Suggestion Change the call to be `totalSupply() - this.totalAmountInCoolDown()`.

2.3 Notes

2.3.1 Sync user amounts on multiple chains

Introduced by `Version 1`

Description In the contract `RemotePancakeIFOHelper`, the function `setUserAmounts()` is invoked by the owner to sync user amounts on multiple chains. Thus, the owner should synchronously invoke the function `setUserAmounts()` on multiple chains as long as the user amounts are changed on one chain.

```
231   function setUserAmounts(
232       address _remotePancakeIFOHelper,
233       address[] calldata _accounts,
234       uint8 _tokenId,
235       uint256[] calldata _amounts
236   ) external onlyOwner {
237       if (_tokenId >= IFOConstantsLib.MAX_TOKEN_NUMBER) revert InvalidTokenId();
238
239       if (_accounts.length > 0) {
240           IRemotePancakeIFOHelper(_remotePancakeIFOHelper).setUserAmounts(_accounts, _tokenId,
                  _amounts);
241       }
242       emit UserAmountsSet();
243   }
```

Listing 2.32: contracts/cakepie/IFO/PancakeIFOHelper.sol

Feedback from the Project Usually, we announce a time and take the snapshot of user stakes at that time. So, we don't sync user balance, every time it changes.

2.3.2 Potential reward dilutions over time

Introduced by `Version 1`

Description In the contract `StreamRewarder`, the rewards are distributed over the time using `duration` and `rewardRate` for calculation. However, when the new rewards queued are less than

the old `rewardRate * duration`, the old rewards will be diluted over the time. For example, at the time of `T1`, when `duration == 100` and old `rewardRate == 20`, the total rewards of `2000` will be accumulated to `rewardPerTokenStored` at `T1 + 100`. However, if `10` rewards queued when `remaining == 50`, the old `2000` of rewards will be diluted over the time to `T1 + 150`. The contract `vlStreamRewarder` also has the feature.

```
243   function _provisionReward(uint256 _rewards, address _rewardToken) internal {
244
245       _rewards = _rewards * DENOMINATOR; // to support small deciaml rewards
246
247       Reward storage rewardInfo = rewards[_rewardToken];
248
249       if (totalStaked() == 0) {
250           rewardInfo.queuedRewards = rewardInfo.queuedRewards + _rewards;
251           return ;
252       }
253
254       rewardInfo.rewardPerTokenStored = rewardPerToken(_rewardToken);
255       _rewards = _rewards + rewardInfo.queuedRewards;
256       rewardInfo.queuedRewards = 0;
257
258       if (block.timestamp >= rewardInfo.periodFinish) {
259           rewardInfo.rewardRate = _rewards / duration;
260       } else {
261           uint256 remaining = rewardInfo.periodFinish - block.timestamp;
262           uint256 leftover = remaining * rewardInfo.rewardRate;
263           _rewards = _rewards + leftover;
264           rewardInfo.rewardRate = _rewards / duration;
265       }
266       rewardInfo.lastUpdateTime = block.timestamp;
267       rewardInfo.periodFinish = block.timestamp + duration;
268
269   }
270
271();
272   }
```

Listing 2.33: contracts/cakepie/rewards/StreamRewarder.sol

2.3.3 Status of mapping `cakeRewardToMcake`

Introduced by `Version 1`

Description Currently, when users claim their own rewards from the contracts `mCakeSV`, `PancakeAMLHelper` and `VLCakepie`, the process of claiming rewards will only invoke the function `multiclaimFor()`, which sets the `_minRecMCake` to be zero. Note that `CakeRewardToMCake` is set true only for `Cake-mCake` Pool. Thus, the Cake rewards will not be converted to mCake rewards except for `Cake-mCake` Pool.

2.3.4 Harvest pools daily through automation

Introduced by Version 1

Description Slippage protection for Cake-to-mCake conversions is applied in the harvest functions, while not applied for deposits and withdrawals. This is because pools are harvested daily through automation and the lower transaction of deposits and withdrawals' volumes make inter-day conversions negligible.

2.3.5 The design of maximum user cap

Introduced by Version 1

Description In the contract PancakeIFOHelper, the function getMaxUserCap() calculates the maximum amount of deposit tokens that users can deposit. The calculation consists of two parts. The first part is getUserCapForMCakeSV(), which is determined by the amount that users locked in the contract mCakeSV. The second part is getUserCapForMCakeLP(), which is determined by the amount of mCakeLpToken that users deposit in the contract PancakeIFOHelper. However, when there are more than one instance of PancakeIFOHelper, users need to deposit mCakeLpToken in every instance of PancakeIFOHelper. In the meantime, users only need to lock once in the mCakeSV.

Feedback from the Project Each IFO has its own and only one PancakeIFOHelper contract. For each IFO, user quota is determined by:

1. user locked balance of mCakeSV at that time
2. user donated balance of mCakeLp for that IFO

2.3.6 Potential centralization risk

Introduced by Version 1

Description The protocol Cakepie has a lot of privileged functions can be operated by the authorized addresses. If the authorized addresses' private key is lost or compromised, it could lead to losses for the protocol and users.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS