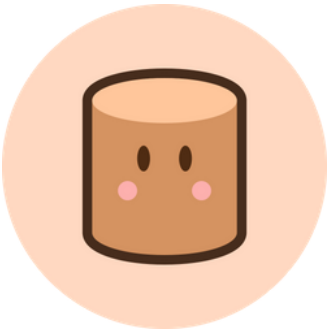
# SweetVaults

## Smart Contract Audit Report
## Prepared for Pacoca



| | |
|---|---|
| **Date Issued:** | Jul 30, 2021 |
| **Project ID:** | AUDIT2021011 |
| **Version:** | v1.0 |
| **Confidentiality Level:** | Public |

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

## Report Information

| | |
|---|---|
| **Project ID** | AUDIT2021011 |
| **Version** | v1.0 |
| **Client** | Pacoca |
| **Project** | SweetVaults |
| **Auditor(s)** | Weerawat Pawanawiwat<br>Suvicha Buakhom<br>Patipon Suwanbol |
| **Author** | Weerawat Pawanawiwat |
| **Reviewer** | Pongsakorn Sommalai |
| **Confidentiality Level** | Public |

## Version History

| Version | Date | Description | Author(s) |
|---|---|---|---|
| 1.0 | Jul 30, 2021 | Full report | Weerawat Pawanawiwat |

## Contact Information



| | |
|---|---|
| **Company** | Inspex |
| **Phone** | (+66) 90 888 7186 |
| **Telegram** | t.me/inspexco |
| **Email** | audit@inspex.co |

# Table of Contents

# 1. Executive Summary

As requested by Pacoca, Inspex team conducted an audit to verify the security posture of the SweetVaults smart contracts on Jul 29, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of SweetVaults smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

## 1.1. Audit Result

In the initial audit, Inspex found 1 high, 1 medium, 4 low, and 1 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved or clarified in the reassessment. Therefore, Inspex trusts that SweetVaults smart contracts have high-level protections in place to be safe from most attacks.


## 1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inpex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

# 2. Project Overview

## 2.1. Project Introduction

Pacoca is a DeFi asset and income monitoring platform on the Binance Smart Chain. The platform provides smart vaults to integrate with other DeFi platforms for compounding yields, and the users can trade tokens on the platform using the integrated decentralized exchange aggregator.

SweetVaults is a new feature of Pacoca that integrates with other DeFi platforms for yield farming. The rewards from the farm are converted to $PACOCA and staked inside Pacoca vault to compound the $PACOCA rewards.

**Scope Information:**

| | |
|---|---|
| **Project Name** | SweetVaults |
| **Website** | https://pacoca.io |
| **Smart Contract Type** | Ethereum Smart Contract |
| **Chain** | Binance Smart Chain |
| **Programming Language** | Solidity |

**Audit Information:**

| | |
|---|---|
| **Audit Method** | Whitebox |
| **Audit Date** | Jul 29, 2021 |
| **Reassessment Date** | Jul 30, 2021 |

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

## 2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

**Initial Audit: (Commit: a8de2ee5b0f1df8a2ab12b2f26bf1c6584f9515a)**

| Contract | Location (URL) |
| --- | --- |
| SweetVault | https://github.com/Pacoca-io/contracts/blob/a8de2ee5b0f1df8a2ab12b2f26bf1c6584f9515a/SweetVault.sol |

**Reassessment: (Commit: 08e9b0bac5d32364c351afac7b7a907785ec1d7f)**

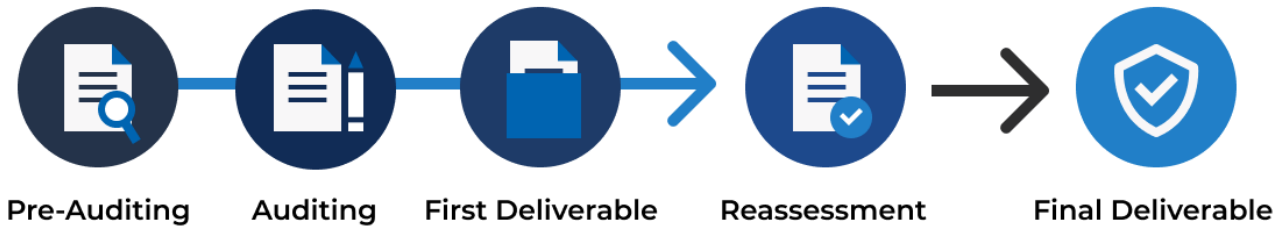
| Contract | Location (URL) |
| --- | --- |
| SweetVault | https://github.com/Pacoca-io/contracts/blob/08e9b0bac5d32364c351afac7b7a907785ec1d7f/SweetVault.sol |

The assessment scope covers only the in-scope smart contracts and the smart contracts that they are inherited from.

# 3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue




## 3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

## 3.2. Audit Items

The following audit items were checked during the auditing activity.

| General |
| --- |
| Reentrancy Attack |
| Integer Overflows and Underflows |
| Unchecked Return Values for Low-Level Calls |
| Bad Randomness |
| Transaction Ordering Dependence |
| Time Manipulation |
| Short Address Attack |
| Outdated Compiler Version |
| Use of Known Vulnerable Component |
| Deprecated Solidity Features |
| Use of Deprecated Component |
| Loop with High Gas Consumption |
| Unauthorized Self-destruct |
| Redundant Fallback Function |
| **Advanced** |
| Business Logic Flaw |
| Ownership Takeover |
| Broken Access Control |
| Broken Authentication |
| Upgradable Without Timelock |
| Improper Kill-Switch Mechanism |
| Improper Front-end Integration |
| Insecure Smart Contract Initiation |

| |
|---|
| Denial of Service |
| Improper Oracle Usage |
| Memory Corruption |
| **Best Practice** |
| Use of Variadic Byte Array |
| Implicit Compiler Version |
| Implicit Visibility Level |
| Implicit Type Inference |
| Function Declaration Inconsistency |
| Token API Violation |
| Best Practices Violation |

## 3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

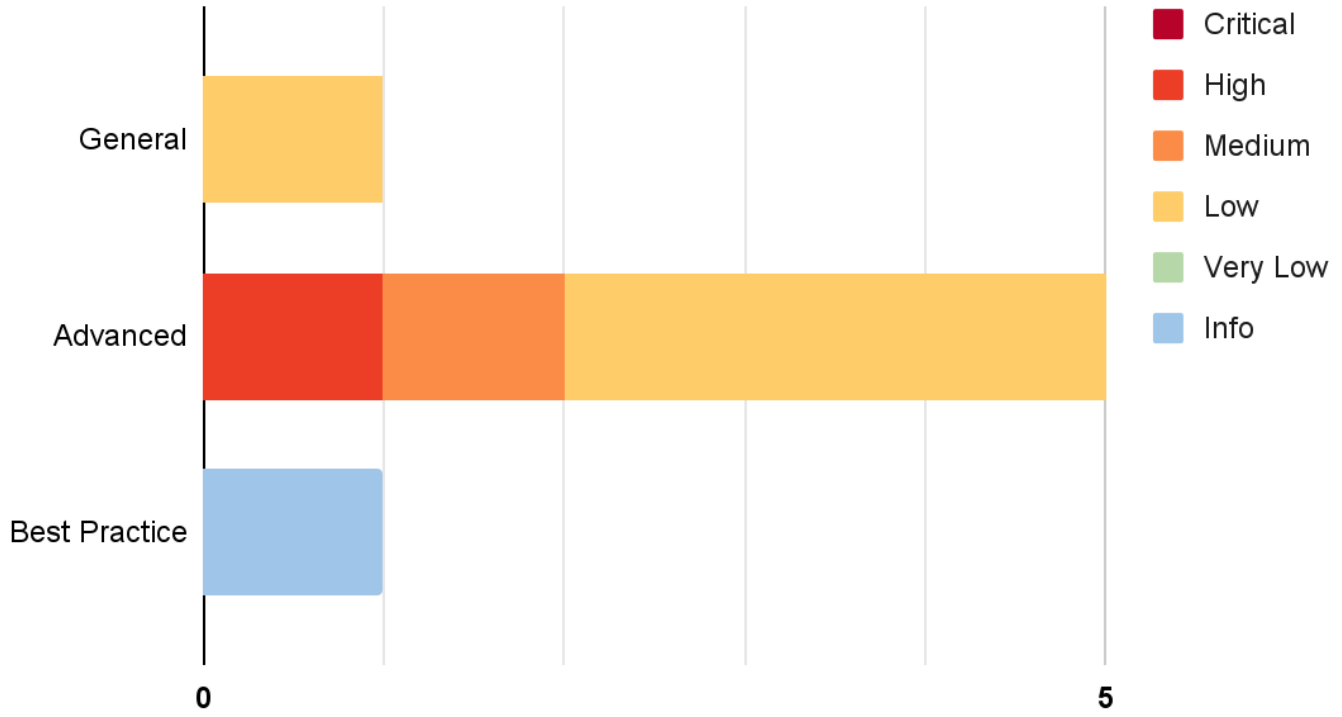
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

**Severity** is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

| Impact \ Likelihood | Low | Medium | High |
|---|---|---|---|
| **Low** | **Very Low** | **Low** | **Medium** |
| **Medium** | **Low** | **Medium** | **High** |
| **High** | **Medium** | **High** | **Critical** |

# 4. Summary of Findings

From the assessments, Inspex has found 7 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.




The statuses of the issues are defined as follows:

| Status | Description |
|---|---|
| Resolved | The issue has been resolved and has no further complications. |
| Resolved * | The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5. |
| Acknowledged | The issue's risk has been acknowledged and accepted. |
| No Security Impact | The best practice recommendation has been acknowledged. |

The information and status of each issue can be found in the following table:

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| IDX-001 | Reward Token Draining by Owner | Advanced | **High** | Resolved |
| IDX-002 | Improper Usage of SafeERC20.safeApprove() | Advanced | **Medium** | Resolved |
| IDX-003 | Unchecked Swapping Path Setting | Advanced | **Low** | Resolved |
| IDX-004 | Improper Reward Transfer Amount Calculation | Advanced | **Low** | Resolved |
| IDX-005 | Dangerous Approval to External Contract | Advanced | **Low** | Resolved |
| IDX-006 | Potential Centralized Control of State Variable | General | **Low** | Resolved * |
| IDX-007 | Improper Function Visibility | Best Practice | **Info** | Resolved |

* The mitigations or clarifications by Pacoca can be found in Chapter 5.

# 5. Detailed Findings Information

## 5.1. Reward Token Draining by Owner

| | |
|---|---|
| **ID** | IDX-001 |
| **Target** | SweetVault |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-284: Improper Access Control |
| **Risk** | **Severity:** High<br><br>**Impact:** Medium<br>The contract owner can drain the reward from the `SweetVault` contract.<br><br>**Likelihood:** High<br>The owner can freely manipulate the router and steal the reward tokens without any restriction. |
| **Status** | Resolved<br>Pacoca team has resolved this issue as suggested in commit 238a3bb1eb97b72fcf46ce493ff6abf7e03857f7. |

### 5.1.1. Description

The `earn()` function in the `SweetVault` contract is used to harvest reward tokens from the farming pool (`STAKED_TOKEN_FARM`).

**SweetVault.sol**

```
125 function earn(uint256 _minPacocaOutput) external onlyKeeper {
126     if (IS_CAKE_STAKING) {
127         STAKED_TOKEN_FARM.leaveStaking(0);
128     } else {
129         STAKED_TOKEN_FARM.withdraw(FARM_PID, 0);
130     }
131
132     _convertBalanceToPacoca(_minPacocaOutput);
133
134     uint256 balanceBeforeFees = _pacocaBalance();
135
136     _safePACOCATransfer(
137         BURN_ADDRESS,
138         balanceBeforeFees.mul(buyBackRate).div(10000)
139     );
140
141     _safePACOCATransfer(
```

```
142             treasury,
143             balanceBeforeFees.mul(keeperFee).div(10000)
144         );
145
146         uint256 previousShares = totalAutoPacocaShares();
147         uint256 pacocaBalance = _pacocaBalance();
148
149         _approveTokenIfNeeded(
150             PACOCA,
151             pacocaBalance,
152             address(AUTO_PACOCA)
153         );
154
155         AUTO_PACOCA.deposit(pacocaBalance);
156
157         uint256 currentShares = totalAutoPacocaShares();
158
159         accSharesPerStakedToken = accSharesPerStakedToken.add(
160             currentShares.sub(previousShares).mul(1e18).div(totalStake())
161         );
162     }
```

After that, the harvested reward tokens will be converted to $PACOCA in the `_convertBalanceToPacoca()` function.

To convert, the `SweetVault` contract allows the `router` to spend the harvested reward tokens (`FARM_REWARD_TOKEN`) by the `_approveTokenIfNeeded()` function and swap it to $PACOCA within the `router.swapExactTokensForTokens()` function.

**SweetVault.sol**

```
339 function _convertBalanceToPacoca(uint256 _minPacocaOutput) private {
340     uint256 rewardTokenBalance = _rewardTokenBalance();
341
342     _approveTokenIfNeeded(
343         FARM_REWARD_TOKEN,
344         rewardTokenBalance,
345         address(router)
346     );
347
348     router.swapExactTokensForTokens(
349         rewardTokenBalance, // input amount
350         _minPacocaOutput,
351         path, // path
352         address(this), // to
353         block.timestamp // deadline
354     );
```

```
355 }
```

However, the contract owner can set the `router` to be any contract in the `setRouter()` function, allowing that `router` to spend the `rewardTokenBalance` freely, for example, sending the `rewardTokenBalance` to a specific wallet, since it is already approved.

**SweetVault.sol**

```
357 function setRouter(address _router) public onlyOwner {
358     address oldRouter = address(router);
359
360     router = IPancakeRouter01(_router);
361
362     emit SetRouter(oldRouter, _router);
363 }
```

The following example describes how the owner could implement the malicious contract and drain all the harvested reward tokens.

**FakeRouter.sol**

```
1  contract FakeRouter {
2      using SafeMath for uint256;
3      using SafeERC20 for IERC20;
4
5      IERC20 constant public PACOCA =
   IERC20(0x55671114d774ee99D653D6C12460c780a67f1D18);
6
7      function swapTokensForExactTokens(
8          uint256 amountIn,
9          uint256 amountOutMin,
10         address[] calldata path,
11         address to,
12         uint256 deadline
13     ) external returns (uint256[] memory amounts) {
14         amounts = [0];
15         PACOCA.safeTransferFrom(msg.sender, address(this), amountIn);
16     }
17
18     ...
19 }
```

### 5.1.2. Remediation

Inspex suggests removing the `setRouter()` function from the `SweetVault` contract. This is because the `router` address can be set from the constructor.

However, if modifications are needed, it is recommended to limit the use of this function via the following options:

- Implementing a community-run governance to control the use of this function
- Using a Timelock contract to delay the changes for a sufficient amount of time

## 5.2. Improper Usage of SafeERC20.safeApprove()

| | |
|---|---|
| **ID** | IDX-002 |
| **Target** | SweetVault |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-840: Business Logic Errors |
| **Risk** | **Severity:** Medium<br><br>**Impact:** Medium<br>Some functions will be unusable due to an unfulfilled condition.<br><br>**Likelihood:** Medium<br>It is likely that the approval amount will eventually be depleted. |
| **Status** | Resolved<br>Pacoca team has resolved this issue in commit 5b64ea548c8e596cbc5f0aa889a21807f3731468 by using `safeIncreaseAllowance()` instead of `safeApprove()` function in the `_approveTokenIfNeeded()` function and approve for just the amount required. |

### 5.2.1. Description

The `_approveTokenIfNeeded()` function uses OpenZeppelin's `SafeERC20.safeApprove()` function to set the allowance of the token when the allowance amount is insufficient.

**SweetVault.sol**

```
300 function _approveTokenIfNeeded(
301     IERC20 _token,
302     uint256 _amount,
303     address _spender
304 ) private {
305     if (_token.allowance(address(this), _spender) < _amount) {
306         _token.safeApprove(_spender, uint(- 1));
307     }
308 }
```

However, the `safeApprove()` function cannot be used to set the allowance from non-zero to non-zero amount.

**@openzeppelin/contracts/token/ERC20/SafeERC20.sol**

```
37 function safeApprove(IERC20 token, address spender, uint256 value) internal {
38     // safeApprove should only be called when setting an initial allowance,
39     // or when resetting it to zero. To increase and decrease it, use
```

```
40      // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
41      // solhint-disable-next-line max-line-length
42      require((value == 0) || (token.allowance(address(this), spender) == 0),
43          "SafeERC20: approve from non-zero to non-zero allowance"
44      );
45      _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector,
    spender, value));
46  }
```

Therefore, the transaction with this function call will be reverted when the allowance is reduced until it is not enough for the transaction, causing some functions to be unusable due to the unfulfilled condition.

### 5.2.2. Remediation

Inspex suggests changing the `safeApprove()` function to `safeIncreaseAllowance()` function with the amount of allowance required, for example:

**SweetVault.sol**

```
300 function _approveTokenIfNeeded(
301     IERC20 _token,
302     uint256 _amount,
303     address _spender
304 ) private {
305     if (_token.allowance(address(this), _spender) < _amount) {
306         _token.safeIncreaseAllowance(_spender, _amount);
307     }
308 }
```

## 5.3. Unchecked Swapping Path Setting

| | |
|---|---|
| **ID** | IDX-003 |
| **Target** | SweetVault |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-755: Improper Handling of Exceptional Conditions |
| **Risk** | **Severity:** Low<br><br>**Impact:** Medium<br>Unchecked path can cause the reward token to be incorrectly swapped, causing the pool to have no reward.<br><br>**Likelihood:** Low<br>There is no direct benefit for the owner to do this, so there is low motivation for this attack. |
| **Status** | Resolved<br>Pacoca team has resolved this issue as suggested in commit 782640a15aca8e9dfc81b5f9313ec017a1fe5a0d. |

### 5.3.1. Description

In the `SweetVault` contract, the `earn()` function is used to harvest the reward from the farm contract. The reward token is then swapped to $PACOCA using the `_convertBalanceToPacoca()` function, which is then used to stake into the Pacoca vault for additional rewards.

**SweetVault.sol**

```
125 function earn(uint256 _minPacocaOutput) external onlyKeeper {
126     if (IS_CAKE_STAKING) {
127         STAKED_TOKEN_FARM.leaveStaking(0);
128     } else {
129         STAKED_TOKEN_FARM.withdraw(FARM_PID, 0);
130     }
131
132     _convertBalanceToPacoca(_minPacocaOutput);
133
134     uint256 balanceBeforeFees = _pacocaBalance();
135
136     _safePACOCATransfer(
137         BURN_ADDRESS,
138         balanceBeforeFees.mul(buyBackRate).div(10000)
139     );
140
141     _safePACOCATransfer(
142         treasury,
```

```
143             balanceBeforeFees.mul(keeperFee).div(10000)
144         );
145
146         uint256 previousShares = totalAutoPacocaShares();
147         uint256 pacocaBalance = _pacocaBalance();
148
149         _approveTokenIfNeeded(
150             PACOCA,
151             pacocaBalance,
152             address(AUTO_PACOCA)
153         );
154
155         AUTO_PACOCA.deposit(pacocaBalance);
156
157         uint256 currentShares = totalAutoPacocaShares();
158
159         accSharesPerStakedToken = accSharesPerStakedToken.add(
160             currentShares.sub(previousShares).mul(1e18).div(totalStake())
161         );
162     }
```

The `path` variable is used to determine the swapping path to convert the reward token to $PACOCA in the `_convertBalanceToPacoca()` function.

**SweetVault.sol**

```
339 function _convertBalanceToPacoca(uint256 _minPacocaOutput) private {
340     uint256 rewardTokenBalance = _rewardTokenBalance();
341
342     _approveTokenIfNeeded(
343         FARM_REWARD_TOKEN,
344         rewardTokenBalance,
345         address(router)
346     );
347
348     router.swapExactTokensForTokens(
349         rewardTokenBalance, // input amount
350         _minPacocaOutput,
351         path, // path
352         address(this), // to
353         block.timestamp // deadline
354     );
355 }
```

The `path` variable can be set by the owner using the `setPath()` function.

**SweetVault.sol**

```
365 function setPath(address[] memory _path) public onlyOwner {
366     address[] memory oldPath = path;
367
368     path = _path;
369
370     emit SetPath(oldPath, path);
371 }
```

However, there is no checking done on the value of the `path`, allowing the swapping path to be set arbitrarily.

An incorrectly set path can cause the pool to have no reward if the swapping destination is not $PACOCA, as there will be no token to stake into the Pacoca vault.

### 5.3.2. Remediation

Inspex suggests checking the source token and destination token in the `setPath()` function, for example:

**SweetVault.sol**

```
365 function setPath(address[] memory _path) public onlyOwner {
366     require(_path[0] == address(FARM_REWARD_TOKEN) && _path[_path.length-1] ==
    address(PACOCA), "Incorrect path");
367     address[] memory oldPath = path;
368
369     path = _path;
370
371     emit SetPath(oldPath, path);
372 }
```

The constructor should also check the source token and destination token in the path, for example:

**SweetVault.sol**

```
85  constructor(
86      address _autoPacoca,
87      address _stakedToken,
88      address _stakedTokenFarm,
89      address _farmRewardToken,
90      uint256 _farmPid,
91      bool _isCakeStaking,
92      address _router,
93      address[] memory _path,
94      address _owner,
95      address _treasury,
96      address _keeper
97  ) public {
```

```
98      require(_path[0] == address(_farmRewardToken) && _path[_path.length - 1] ==
    address(PACOCA), "Incorrect path");
99      AUTO_PACOCA = IPacocaVault(_autoPacoca);
100     STAKED_TOKEN = IERC20(_stakedToken);
101     STAKED_TOKEN_FARM = IPancakeswapFarm(_stakedTokenFarm);
102     FARM_REWARD_TOKEN = IERC20(_farmRewardToken);
103     FARM_PID = _farmPid;
104     IS_CAKE_STAKING = _isCakeStaking;
105
106     router = IPancakeRouter01(_router);
107     path = _path;
108
109     transferOwnership(_owner);
110     treasury = _treasury;
111     keeper = _keeper;
112 }
```

## 5.4. Improper Reward Transfer Amount Calculation

| | |
|---|---|
| **ID** | IDX-004 |
| **Target** | SweetVault |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-840: Business Logic Errors |
| **Risk** | **Severity:** Low<br><br>**Impact:** Medium<br>The users may gain a higher amount of $PACOCA than the actual amount of reward claimed.<br><br>**Likelihood:** Low<br>It is unlikely that there will be $PACOCA left inside the `SweetVault` contract. |
| **Status** | Resolved<br>Pacoca team has resolved this issue as suggested in commit ade4cd236e70473dbdf470ebe85b82d6afc02efe and confirmed that `SweetVault` will be used with the vault without withdrawal fee. |

### 5.4.1. Description

The `_claimReward()` function can be used to withdraw the user's portion of $PACOCA reward from the Pacoca vault. The amount of share to withdraw is the input of the function, and the amount of $PACOCA to be transferred to the user is calculated by multiplying the price per share with the amount of share in line 263. The share is withdrawn from the vault as $PACOCA in line 265, and the $PACOCA is then transferred to the user using the `_safePACOCATransfer()` function in line 267.

**SweetVault.sol**

```
245 function _claimRewards(uint256 _shares, bool _update) private {
246     UserInfo storage user = userInfo[msg.sender];
247
248     if (_update) {
249         user.autoPacocaShares = user.autoPacocaShares.add(
250             user.stake.mul(accSharesPerStakedToken).div(1e18).sub(
251                 user.rewardDebt
252             )
253         );
254
255         user.rewardDebt = user.stake.mul(accSharesPerStakedToken).div(1e18);
256     }
257
258     require(user.autoPacocaShares >= _shares, "SweetVault: claim amount exceeds
```

```
balance");
259
260         user.autoPacocaShares = user.autoPacocaShares.sub(_shares);
261
262         uint256 pricePerShare = AUTO_PACOCA.getPricePerFullShare();
263         uint256 pacocaAmount = _shares.mul(pricePerShare).div(1e18);
264
265         AUTO_PACOCA.withdraw(_shares);
266
267         _safePACOCATransfer(msg.sender, pacocaAmount);
268
269         emit ClaimRewards(msg.sender, _shares, pacocaAmount);
270     }
```

However, if the Pacoca vault collects the withdrawal fee from the user, the actual amount withdrawn from the Pacoca vault can be less than the amount calculated (`pacocaAmount`). As the `_safePACOCATransfer()` function handles the case when there is insufficient $PACOCA for the transfer, the transfer in not reverted, but if there is leftover $PACOCA inside the contract, the user will gain more $PACOCA than the actual amount claimed, and the fee will be imposed on the `SweetVault` contract.

**SweetVault.sol**

```
329 function _safePACOCATransfer(address _to, uint256 _amount) private {
330     uint256 balance = _pacocaBalance();
331
332     if (_amount > balance) {
333         PACOCA.transfer(_to, balance);
334     } else {
335         PACOCA.transfer(_to, _amount);
336     }
337 }
```

The platform users should note that if the Pacoca vault collects the withdraw fee, the contract may potentially stay within the withdraw fee period most of the time due to the calling of `deposit()` and `earn()` functions that deposit to the Pacoca vault, and withdraw fee may be imposed to most withdraw transactions in `SweetVault`.

### 5.4.2. Remediation

Inpex suggests checking the balance and transfer the actual amount of $PACOCA withdrawn to the contract, for example:

**SweetVault.sol**

```
245 function _claimRewards(uint256 _shares, bool _update) private {
246     UserInfo storage user = userInfo[msg.sender];
247 
248     if (_update) {
249         user.autoPacocaShares = user.autoPacocaShares.add(
250             user.stake.mul(accSharesPerStakedToken).div(1e18).sub(
251                 user.rewardDebt
252             )
253         );
254 
255         user.rewardDebt = user.stake.mul(accSharesPerStakedToken).div(1e18);
256     }
257 
258     require(user.autoPacocaShares >= _shares, "SweetVault: claim amount exceeds
    balance");
259 
260     user.autoPacocaShares = user.autoPacocaShares.sub(_shares);
261 
262     uint256 pacocaBalanceBefore = _pacocaBalance();
263 
264     AUTO_PACOCA.withdraw(_shares);
265 
266     uint256 pacocaAmount = _pacocaBalance().sub(pacocaBalanceBefore);
267 
268     _safePACOCATransfer(msg.sender, pacocaAmount);
269 
270     emit ClaimRewards(msg.sender, _shares, pacocaAmount);
271 }
```

## 5.5. Dangerous Approval to External Contract

| | |
|---|---|
| **ID** | IDX-005 |
| **Target** | SweetVault |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-284: Improper Access Control |
| **Risk** | **Severity:** Low<br><br>**Impact:** Low<br>The external contracts can steal all approved tokens from the `SweetVault` contract; however, for normal usage, no token will be left unattended in the `SweetVault` contract.<br><br>**Likelihood:** Medium<br>It is not likely that the external contracts specifically defined by the owner will steal tokens from the `SweetVault` contract. |
| **Status** | Resolved<br>Pacoca team has resolved this issue in commit 5b64ea548c8e596cbc5f0aa889a21807f3731468 by using `safeIncreaseAllowance()` instead of `safeApprove()` function in the `_approveTokenIfNeeded()` function and approve for just the amount required. |

### 5.5.1. Description

The `_approveTokenIfNeeded()` function is used in multiple functions of `SweetVault` contract to allow other contracts to spend the tokens inside the `SweetVault` contract, for example:

**SweetVault.sol**

```
339 function _convertBalanceToPacoca(uint256 _minPacocaOutput) private {
340     uint256 rewardTokenBalance = _rewardTokenBalance();
341
342     _approveTokenIfNeeded(
343         FARM_REWARD_TOKEN,
344         rewardTokenBalance,
345         address(router)
346     );
347
348     router.swapExactTokensForTokens(
349         rewardTokenBalance, // input amount
350         _minPacocaOutput,
351         path, // path
352         address(this), // to
353         block.timestamp // deadline
354     );
```

```
355 }
```

However, the approval amount is set to `uint(- 1)`, which is the maximum of `uint`, instead of approving only the required amount.

**SweetVault.sol**

```
300 function _approveTokenIfNeeded(
301     IERC20 _token,
302     uint256 _amount,
303     address _spender
304 ) private {
305     if (_token.allowance(address(this), _spender) < _amount) {
306         _token.safeApprove(_spender, uint(- 1));
307     }
308 }
```

This allows external contracts to freely transfer tokens from the `SweetVault` contract. However, for normal usage, no token will be left unattended in the `SweetVault` contract.

The `_approveTokenIfNeeded()` function is used in the following locations:

| Function | Line |
|---|---|
| earn() | 149 |
| deposit() | 175 |
| _convertBalanceToPacoca() | 342 |

### 5.5.2. Remediation

Inspex suggests removing the `_approveTokenIfNeeded()` function and replacing all usages of it. Approval should be done to approve only the necessary amount of allowance, and then revoke after the process has finished, for example:

**SweetVault.sol**

```
339 function _convertBalanceToPacoca(uint256 _minPacocaOutput) private {
340     uint256 rewardTokenBalance = _rewardTokenBalance();
341 
342     FARM_REWARD_TOKEN.safeApprove(address(router), rewardTokenBalance);
343 
344     router.swapExactTokensForTokens(
345         rewardTokenBalance, // input amount
346         _minPacocaOutput,
347         path, // path
348         address(this), // to
349         block.timestamp // deadline
350     );
351 
352     FARM_REWARD_TOKEN.safeApprove(address(router), 0);
353 }
```

## 5.6. Potential Centralized Control of State Variable

| | |
|---|---|
| **ID** | IDX-006 |
| **Target** | SweetVault |
| **Category** | General Smart Contract Vulnerability |
| **CWE** | CWE-710: Improper Adherence to Coding Standards |
| **Risk** | **Severity:** Low<br><br>**Impact:** Low<br>The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the users.<br><br>**Likelihood:** Medium<br>There is potentially nothing to restrict the changes from being done; however, the changes are limited by fixed values in the smart contract. |
| **Status** | Resolved *<br>The smart contract is not yet deployed at the time of reassessment, but the Pacoca team has confirmed that the `Timelock` will be added. |

### 5.6.1. Description

Critical state variables can be updated any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, as the contract is not yet deployed, there is potentially no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

| Target | Function | Modifier |
|---|---|---|
| SweetVault (L:357) | setRouter() | onlyOwner |
| SweetVault (L:365) | setPath() | onlyOwner |
| SweetVault (L:373) | setTreasury() | onlyOwner |
| SweetVault (L:381) | setKeeper() | onlyOwner |
| SweetVault (L:389) | setKeeperFee() | onlyOwner |
| SweetVault (L:399) | setBuyBackRate() | onlyOwner |

| | | |
|---|---|---|
| SweetVault (L:412) | setEarlyWithdrawFee() | onlyOwner |
| Ownable | renounceOwnership() | onlyOwner |
| Ownable | transferOwnership() | onlyOwner |

Please note that the `Ownable` contract is inherited from the OpenZeppelin library.

### 5.6.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a Timelock contract to delay the changes for a sufficient amount of time

## 5.7. Improper Function Visibility

| ID | IDX-007 |
|---|---|
| Target | SweetVault |
| Category | Smart Contract Best Practice |
| CWE | CWE-710: Improper Adherence to Coding Standards |
| Risk | **Severity:** Info<br>**Impact:** None<br>**Likelihood:** None |
| Status | Resolved<br>Pacoca team has resolved this issue as suggested in commit 08e9b0bac5d32364c351afac7b7a907785ec1d7f. |

### 5.7.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

For example, the following source code shows that the `claimRewards()` function of the `SweetVault` is set to `public` and it is never called from any internal function.

**SweetVault.sol**

```
241 function claimRewards(uint256 _shares) public nonReentrant {
242     _claimRewards(_shares, true);
243 }
```

The following table contains all functions that have `public` visibility and are never called from any internal function.

| Target | Function |
|---|---|
| SweetVault.sol (L:241) | claimRewards() |
| SweetVault.sol (L:357) | setRouter() |
| SweetVault.sol (L:365) | setPath() |
| SweetVault.sol (L:373) | setTreasury() |
| SweetVault.sol (L:381) | setKeeper() |

| | |
|---|---|
| SweetVault.sol (L:389) | setKeeperFee() |
| SweetVault.sol (L:399) | setBuyBackRate() |
| SweetVault.sol (L:412) | setEarlyWithdrawFee() |

### 5.7.2. Remediation

Inspex suggests changing all functions' visibility to external if they are not called from any internal function as shown in the following example:

**SweetVault.sol**

```
241 function claimRewards(uint256 _shares) external nonReentrant {
242     _claimRewards(_shares, true);
243 }
```

# 6. Appendix

## 6.1. About Inspex


Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

**Follow Us On:**

| | |
|---|---|
| **Website** | https://inspex.co |
| **Twitter** | @InspexCo |
| **Facebook** | https://www.facebook.com/InspexCo |
| **Telegram** | @inspex_announcement |

## 6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE