CakeMaxi

Smart Contract Audit Report
Prepared for Alpaca Finance

Date Issued:	Jun 14, 2021
Project ID:	AUDIT2021002
Version:	v3.0
Confidentiality Level:	Public

inspex

CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021002
Version	3.0
Client	Alpaca Finance
Project	CakeMaxi
Auditor(s)	Weerawat Pawanawiwat Pongsakorn Sommalai Suvicha Buakhom
Author	Weerawat Pawanawiwat
Reviewer	Pongsakorn Sommalai
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
3.0	Jun 14, 2021	Add audit round 2 result	Weerawat Pawanawiwat
2.0	Jun 1, 2021	Add additional responses from the Alpaca Finance team	Weerawat Pawanawiwat
1.0	May 28, 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 Alpaca Finance, Inspex team conducted audits to verify the security posture of the CakeMaxi smart contracts on May 24, 2021, May 25, 2021, and Jun 12, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of CakeMaxi 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 medium, 1 low, 1 very low, and 1 info-severity issues. With the project team's prompt response, 1 medium and 1 info-severity issues were resolved in the reassessment, while 1 low and 1 very low-severity issues were acknowledged by the team. Therefore, Inspex trusts that CakeMaxi smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

This smart contract passes Inspex's security verification standard, and is trustworthy.

Approved by Inspex on: Jun 14, 2021

PASS
inspex
CYBERSECURITY
PROFESSIONAL
SERVICE

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

Alpaca Finance is the largest lending protocol allowing leveraged yield farming on Binance Smart Chain. It helps lenders to earn safe and stable yields, and offers borrowers undercollateralized loans for leveraged yield farming positions, vastly multiplying their farming principals and resulting profits.

CakeMaxi is a new feature for Alpaca Finance, extending the existing features, allowing users to open leveraged yield farming positions on PancakeSwap CAKE Syrup Pool to maximize their $CAKE reward.

Scope Information:

Project Name	CakeMaxi
Website	https://app.alpacafinance.org/
Smart Contract Type	Ethereum Smart Contract
Programming Language	Solidity

Audit Information (Round 1):

Audit Method	Whitebox
Audit Date	May 24, 2021 - May 25, 2021
Reassessment Date	May 26, 2021

Audit Information (Round 2):

Audit Method	Whitebox
Audit Date	Jun 12, 2021

2.2. Scope

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

Initial Audit (Round 1):

Name	Location (URL)
CakeMaxiWorker.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/1f89672f65acdedc9a4852a0f9afce05e31cae75/contracts/6/protocol/workers/CakeMaxiWorker.sol
CakeMaxiWorkerConfig.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/1f89672f65acdedc9a4852a0f9afce05e31cae75/contracts/6/protocol/workers/CakeMaxiWorkerConfig.sol
PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/1f89672f65acdedc9a4852a0f9afce05e31cae75/contracts/6/protocol/strategies/pancakeswapV2-restricted-cake-maxi/PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol
PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/1f89672f65acdedc9a4852a0f9afce05e31cae75/contracts/6/protocol/strategies/pancakeswapV2-restricted-cake-maxi/PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol
PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/1f89672f65acdedc9a4852a0f9afce05e31cae75/contracts/6/protocol/strategies/pancakeswapV2-restricted-cake-maxi/PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol
PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/1f89672f65acdedc9a4852a0f9afce05e31cae75/contracts/6/protocol/strategies/pancakeswapV2-restricted-cake-maxi/PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol

Reassessment (Round 1):

Name	Location (URL)
CakeMaxiWorker.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/9966ba97a8593eda58b013f38d6c60fb50519e06/contracts/6/protocol/workers/CakeMaxiWorker.sol
CakeMaxiWorkerConfig.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/9966ba97a8593eda58b013f38d6c60fb50519e06/contracts/6/protocol/workers/CakeMaxiWorkerConfig.sol
PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/9966ba97a8593eda58b013f38d6c60fb50519e06/contracts/6/protocol/strategies/pancakeswapV2-restricted-cake-maxi/PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol
PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/9966ba97a8593eda58b013f38d6c60fb50519e06/contracts/6/protocol/strategies/pancakeswapV2-restricted-cake-maxi/PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol
PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/9966ba97a8593eda58b013f38d6c60fb50519e06/contracts/6/protocol/strategies/pancakeswapV2-restricted-cake-maxi/PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol
PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/9966ba97a8593eda58b013f38d6c60fb50519e06/contracts/6/protocol/strategies/pancakeswapV2-restricted-cake-maxi/PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol

Initial Audit (Round 2):

Name	Location (URL)
CakeMaxiWorker.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/20c4c545a0323b71d2969fd79db8316e60bc7d76/contracts/6/protocol/workers/CakeMaxiWorker.sol
SingleAssetWorkerConfig.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/20c4c545a0323b71d2969fd79db8316e60bc7d76/contracts/6/protocol/workers/SingleAssetWorkerConfig.sol
PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/20c4c545a0323b71d2969fd79db8316e60bc7d76/contracts/6/protocol/strategies/pancakeswapV2-restricted-single-asset/PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol
PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/20c4c545a0323b71d2969fd79db8316e60bc7d76/contracts/6/protocol/strategies/pancakeswapV2-restricted-single-asset/PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol
PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/20c4c545a0323b71d2969fd79db8316e60bc7d76/contracts/6/protocol/strategies/pancakeswapV2-restricted-single-asset/PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol
PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol	https://github.com/alpaca-finance/bsc-alpaca-contract/blob/20c4c545a0323b71d2969fd79db8316e60bc7d76/contracts/6/protocol/strategies/pancakeswapV2-restricted-single-asset/PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol

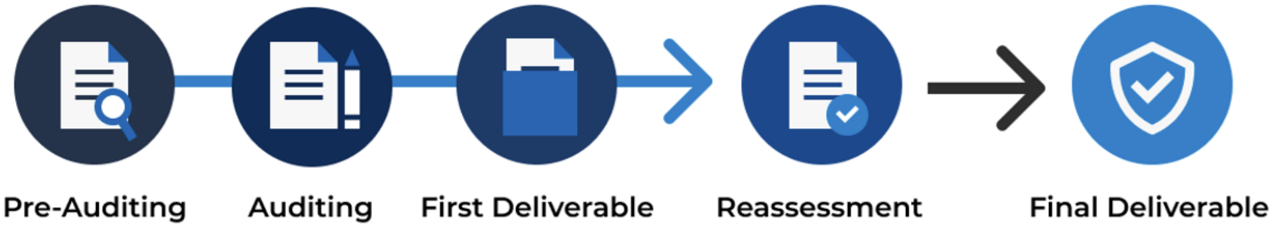
Reassessment (Round 2):

There is no issue that needed the reassessment activity.

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

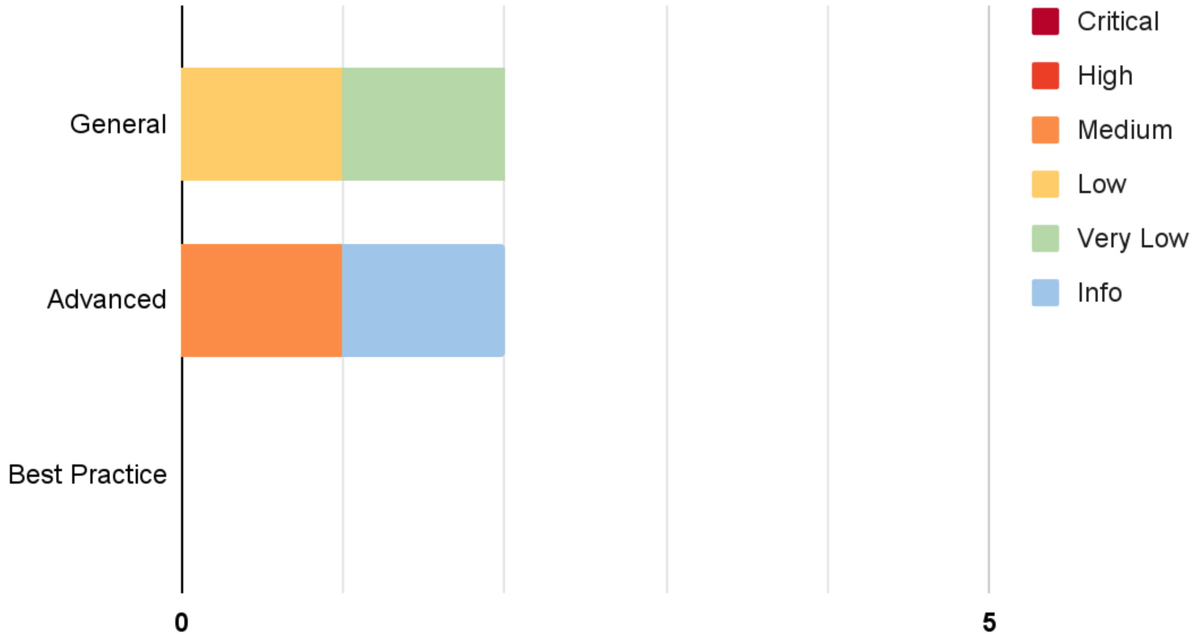
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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**.

Likelihood Impact	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 4 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.
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:

Audit Round 1

ID	Title	Category	Severity	Status
IDX-001	Improper Swapping Tolerance Calculation	Advanced	**Medium**	Resolved
IDX-002	Transaction Ordering Dependence	General	**Low**	Acknowledged
IDX-003	Outdated Solidity Version	General	**Very Low**	Acknowledged
IDX-004	Potential Upgradable Without Timelock	Advanced	**Info**	Resolved *

Audit Round 2

There is no additional issue found during the assessment.

5. Detailed Findings Information

5.1. Improper Swapping Tolerance Calculation

ID	IDX-001
Scope	Audit Round 1
Target	PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The price impact can be higher than the tolerance value, causing the position to lose value from a bad swapping rate. **Likelihood:** Medium It is likely that other transactions can be done in the same pool during the period between the price estimation and the strategy execution. However, if the amount of `inputFarmingTokenAmount` is not large, there is still some tolerance to prevent drastic price change.
Status	Resolved The Alpaca Finance team has resolved this issue.

5.1.1. Description

The `PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm` strategy contract can be used to add both `baseToken` together with `farmingToken` into a farming position. In the `execute()` function, the `data` parameter is decoded into two variables, `inputFarmingTokenAmount` and `minFarmingTokenAmount`.

`inputFarmingTokenAmount` is used to determine the amount of `farmingToken` to be transferred to the strategy contract by `vault.requestFunds()` function.

PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol

```
function execute(address /* user */, uint256 /* debt */, bytes calldata data)
  external
  override
  onlyWhitelistedWorkers
  nonReentrant
{
```

```
52    // 1. Find out how many farmingToken amount the strategy should deal with and min additional farmingTokens.
53    (
54      uint256 inputFarmingTokenAmount,
55      uint256 minFarmingTokenAmount
56    ) = abi.decode(data, (uint256, uint256));
57    IWorker worker = IWorker(msg.sender);
58    address baseToken = worker.baseToken();
59    address farmingToken = worker.farmingToken();
60    // 2. Approve router to do their stuffs
61    baseToken.safeApprove(address(router), uint256(-1));
62    // 3. request additional fund in form of a farmingToken from the vault using inputFarmingTokenAmount
63    vault.requestFunds(farmingToken, inputFarmingTokenAmount);
```

`minFarmingTokenAmount` is used as the price impact tolerance threshold, making sure that the `farmingToken` gained from swapping the `baseToken` must be more than or equal to that value. In line 82, the `amountOutMin` parameter of `swapExactTokensForTokens()` functions is set to 0, meaning that the price impact tolerance for this function call is unlimited. The actual amount swapped is then checked in line 83; however, the initial amount of `farmingToken` from `inputFarmingTokenAmount` is not considered, so the `minFarmingTokenAmount` is compared with the total balance in the contract, not the actual amount swapped.

PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol

```
81    router.swapExactTokensForTokens(balance, 0, path, address(this), now);
82    // 5. Transfer all farming token (as a result of conversion) back to the calling worker
83    require(farmingToken.myBalance() >= minFarmingTokenAmount, "PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm::execute:: insufficient farmingToken amount received");
84    farmingToken.safeTransfer(msg.sender, farmingToken.myBalance());
85    // 6. Reset approval for safety reason
86    baseToken.safeApprove(address(router), 0);
87  }
```

5.1.2. Remediation

Inspex suggests comparing `minFarmingTokenAmount` to the actual amount of `farmingToken` resulting from the swap. This can be done by taking the `inputFarmingTokenAmount` into consideration during the comparison. The example fix is shown in the code snippet below:

PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol

```
  router.swapExactTokensForTokens(balance, 0, path, address(this), now);
  // 5. Transfer all farming token (as a result of conversion) back to the calling worker
  require(farmingToken.myBalance().sub(inputFarmingTokenAmount) >= minFarmingTokenAmount, "PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm::execute:: insufficient farmingToken amount received");
  farmingToken.safeTransfer(msg.sender, farmingToken.myBalance());
  // 6. Reset approval for safety reason
  baseToken.safeApprove(address(router), 0);
}
```

5.2. Transaction Ordering Dependence

ID	IDX-002
Scope	Audit Round 1
Target	CakeMaxiWorker.sol
Category	General Smart Contract Vulnerability
CWE	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
Risk	**Severity:** Low **Impact:** Medium The Front Running attack can be performed, resulting in a bad swapping rate and a lower bounty. **Likelihood:** Low It is easy to perform the attack. However, with a low profit, there is low motivation to attack with this vulnerability.
Status	**Acknowledged** The Alpaca Finance team has acknowledged the vulnerability. However, the risks are quite low due to the amount of $CAKE that is being reinvested is small compared to the liquidity in $CAKE-$BNB LP Pool.

5.2.1. Description

In Alpaca Finance, to compound the rewards, the reinvestor calls the `reinvest()` function of the `CakeMaxiWorker` contract. The rewards are allocated to the reinvestor and beneficial vault as shown below:

CakeMaxiWorker.sol

```
/// @dev Re-invest whatever this worker has earned to the staking pool.
function reinvest() external override onlyEOA onlyReinvestor nonReentrant {
  // 1. Approve tokens
  farmingToken.safeApprove(address(masterChef), uint256(-1));
  // 2. reset all reward balance since all rewards will be reinvested
  rewardBalance = 0;
  // 3. Withdraw all the rewards.
  masterChef.leaveStaking(0);
  uint256 reward = farmingToken.myBalance();
  if (reward == 0) return;
  // 4. Send the reward bounty to the caller.
  uint256 bounty = reward.mul(reinvestBountyBps) / 10000;
  if (bounty > 0) {
    uint256 beneficialVaultBounty = bounty.mul(beneficialVaultBountyBps) /
10000;
    if (beneficialVaultBounty > 0)
_rewardToBeneficialVault(beneficialVaultBounty, farmingToken);
    farmingToken.safeTransfer(msg.sender, bounty.sub(beneficialVaultBounty));
  }
  // 5. re stake the farming token to get more rewards
  masterChef.enterStaking(reward.sub(bounty));
  // 6. Reset approval
  farmingToken.safeApprove(address(masterChef), 0);
  emit Reinvest(msg.sender, reward, bounty);
}
```

Next, `_rewardToBeneficialVault()` function is called to swap `rewardToken` to the token accepted by the beneficial vault and transfer them to the vault as follows:

CakeMaxiWorker.sol

```
function _rewardToBeneficialVault(uint256 _beneficialVaultBounty, address
_rewardToken) internal {
  _rewardToken.safeApprove(address(router), uint256(-1));
  address beneficialVaultToken = beneficialVault.token();
  address[] memory path = _getPath(_rewardToken, beneficialVaultToken);
  router.swapExactTokensForTokens(_beneficialVaultBounty, 0, path,
address(this), now);
  beneficialVaultToken.safeTransfer(address(beneficialVault),
beneficialVaultToken.myBalance());
  _rewardToken.safeApprove(address(router), 0);
}
```

It can be seen in the above source code, the `router.swapExactTokensForTokens()` function is called by setting the `amountOutMin` to 0. Therefore, the Front Running attack can be performed, resulting in a bad swapping rate and a lower bounty.

5.2.2. Remediation

The tolerance value (amountOutMin) should not be set to 0. Inspex suggests calculating the expected amount out with the token price fetched from the price oracles or passed from the investor bot directly, and setting it to the amountOutMin parameter while calling the router.swapExactTokensForTokens() function as shown in the following example:

CakeMaxiWorker.sol

```
function _rewardToBeneficialVault(uint256 _beneficialVaultBounty, address _rewardToken) internal {
  _rewardToken.safeApprove(address(router), uint256(-1));
  address beneficialVaultToken = beneficialVault.token();
  address[] memory path = _getPath(_rewardToken, beneficialVaultToken);
  uint256 amountOutMin = calcAmountOutMin(_beneficialVaultBounty);
  router.swapExactTokensForTokens(_beneficialVaultBounty, amountOutMin, path, address(this), now);
  beneficialVaultToken.safeTransfer(address(beneficialVault), beneficialVaultToken.myBalance());
  _rewardToken.safeApprove(address(router), 0);
}
```

5.3. Outdated Solidity Version

ID	IDX-003
Scope	Audit Round 1
Target	CakeMaxiWorker.sol CakeMaxiWorkerConfig.sol PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Very Low **Impact:** Low From the list of known Solidity bugs, the direct impact cannot be caused by those bugs themselves. **Likelihood:** Low From the list of known Solidity bugs, it is very unlikely that those bugs would affect these smart contracts.
Status	**Acknowledged** The Alpaca Finance team has acknowledged this issue. The team decided to leave the compiler in 0.6.6 version as known issues have no relation to the flow of the codes and so are highly unlikely to have any impact. All interfaces and library related are all written previously and frozen at 0.6.6, so changing the version could have effect across all 0.6.6 contracts.

5.3.1. Description

The Solidity compiler version specified in the smart contracts was outdated. This version has publicly known inherent bugs[2] that may potentially be used to cause damage to the smart contracts or the users of the smart contracts. The compiler version used in the smart contracts are as follows:

File	Compiler Version
CakeMaxiWorker.sol	0.6.6
CakeMaxiWorkerConfig.sol	0.6.6

PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol	0.6.6
PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol	0.6.6
PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol	0.6.6
PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol	0.6.6

For example, the compiler version is specified at the top of the source code file as follows:

CakeMaxiWorker.sol

```
pragma solidity 0.6.6;
```

5.3.2. Remediation

Inspex suggests upgrading the Solidity compiler to the latest stable version[3].

During the audit activity, the latest stable version of the Solidity compiler in major 0.6 is v0.6.12.

5.4. Potential Upgradable Without Timelock

ID	IDX-004
Scope	Audit Round 1
Target	CakeMaxiWorker.sol CakeMaxiWorkerConfig.sol PancakeswapV2RestrictedCakeMaxiStrategyAddBaseTokenOnly.sol PancakeswapV2RestrictedCakeMaxiStrategyAddBaseWithFarm.sol PancakeswapV2RestrictedCakeMaxiStrategyLiquidate.sol PancakeswapV2RestrictedCakeMaxiStrategyWithdrawMinimizeTrading.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Info **Impact:** None **Likelihood:** None
Status	Resolved * The Alpaca Finance team has planned to delegate the ownership of the contracts to the timelock contract when they are deployed, just like the other core smart contracts of Alpaca Finance Protocol that are already under timelock.

5.4.1. Description

At the time of the audit, the smart contracts were not deployed yet. As these smart contracts are upgradable, if the contracts are not protected using timelock, the logic of the smart contract could be modified by the owner anytime, making the smart contract untrustworthy. The timelock mechanism sets a time delay before any change can be made to the smart contract source code, and will provide time for users to monitor the changes and take action safely before any potentially unwanted change has taken effect.

CakeMaxiWorker.sol

```
contract CakeMaxiWorker is OwnableUpgradeSafe, ReentrancyGuardUpgradeSafe, IWorker {
  /// @notice Libraries
```

5.4.2. Remediation

Inspex recommends deploying the contracts with the timelock mechanism by setting the contract owner to the timelock contract.

6. Appendix

6.1. About Inspex

CYBERSECURITY
PROFESSIONAL
SERVICE

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]

[2] "List of Known Bugs — Solidity 0.6.6 documentation." [Online]. Available: https://docs.soliditylang.org/en/v0.6.6/bugs.html. [Accessed: 27-May-2021]

[3] ethereum, "Releases · ethereum/solidity." [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 27-May-2021]

inspex

CYBERSECURITY PROFESSIONAL SERVICE