Alpaca Finance 2.0 Automated Vaults V3

Smart Contract Audit Report
Prepared for Alpaca Finance

Date Issued:	Aug 4, 2023
Project ID:	AUDIT2023012
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2023012
Version	v1.0
Client	Alpaca Finance
Project	Alpaca Finance 2.0 Automated Vaults V3
Auditor(s)	Peeraphut Punsuwan Ronnachai Chaipha Kongkit Chatchawanhirun
Author(s)	Peeraphut Punsuwan Ronnachai Chaipha Kongkit Chatchawanhirun
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Aug 4, 2023	Full report	Peeraphut Punsuwan Ronnachai Chaipha Kongkit Chatchawanhirun

Contact Information

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

Table of Contents

1. Executive Summary **1**
1.1. Audit Result 1
1.2. Disclaimer 1
2. Project Overview **2**
2.1. Project Introduction 2
2.2. Scope 3
3. Methodology **6**
3.1. Test Categories 6
3.2. Audit Items 7
3.3. Risk Rating 9
4. Summary of Findings **10**
5. Detailed Findings Information **12**
5.1. Centralized Control of State Variable 12
5.2. Denial of Service on Withdrawal 15
5.3. Force Decrease Liquidity on Withdrawal 26
5.4. Transaction Ordering Dependence 31
5.5. PancakeSwap V3 Pair Price Manipulation 35
5.6. Slippage Tolerance Exceed Price Range 39
6. Appendix **44**
6.1. About Inspex 44

1. Executive Summary

As requested by Alpaca Finance, Inspex team conducted an audit to verify the security posture of the Alpaca Finance 2.0 Automated Vaults V3 smart contracts between Jul 18, 2023 and Jul 21, 2023. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Alpaca Finance 2.0 Automated Vaults V3 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 2 high, 1 medium, 2 low, 1 very low-severity issues. With the project team's prompt response, 2 high, 1 medium and 1 very low-severity issues were resolved or mitigated in the reassessment, while 2 low-severity issues were acknowledged by the team. Therefore, Inspex trusts that Alpaca Finance 2.0 Automated Vaults V3 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.

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), Inspex 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 - Automated Vaults V3 is a platform designed for yield farming across various strategies on PancakeSwap V3. By utilizing this platform, investors gain access to automated vaults that handle yield farming tasks such as leverage farming, auto-compounding, and rebalancing, all while mitigating the risk of liquidation. In comparison to the workings of traditional mutual funds, where investors purchase shares and rely on fund managers to make profitable investment decisions behind the scenes, the automated vaults offer a different approach. They aim to create a secure and transparent investment pool, providing a clear view of the investment process and outcomes for the investors.

Scope Information:

Project Name	Alpaca Finance 2.0 Automated Vaults V3
Website	https://www.alpacafinance.org/
Smart Contract Type	Ethereum Smart Contract
Chain	BNB Smart Chain
Programming Language	Solidity
Category	Yield Farming, Auto Compound, Token

Audit Information:

Audit Method	Whitebox
Audit Date	Jul 18, 2023 - Jul 21, 2023
Reassessment Date	Aug 3, 2023

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: 44c9da92226effb4368ce45832909890e538dab5)

Contract	Location (URL)
AutomatedVaultERC20	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/AutomatedVaultERC20.sol
AutomatedVaultManager	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/AutomatedVaultManager.sol
Bank	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/Bank.sol
Executor	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/executors/Executor.sol
PCSV3Executor01	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/executors/PCSV3Executor01.sol
PCSV3StableExecutor	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/executors/PCSV3StableExecutor.sol
AVManagerV3Gateway	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/gateway/AVManagerV3Gateway.sol
Constants	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/libraries/Constants.sol
LibFixedPoint128	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/libraries/LibFixedPoint128.sol
LibFixedPoint96	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/libraries/LibFixedPoint96.sol
LibFullMath	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/libraries/LibFullMath.sol
LibLiquidityAmounts	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/libraries/LibLiquidityAmounts.sol
LibShareUtil	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/libraries/LibShareUtil.sol
LibSqrtPriceX96	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/libraries/LibSqrtPriceX96.sol

LibTickMath	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/libraries/LibTickMath.sol
BaseOracle	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/oracles/BaseOracle.sol
PancakeV3VaultOracle	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/oracles/PancakeV3VaultOracle.sol
PancakeV3VaultReader	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/reader/PancakeV3VaultReader.sol
PancakeV3Worker	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/44c9da9222/src/workers/PancakeV3Worker.sol

Reassessment: (Commit: 64711d1464b08ffc47483f587f60a5387690fe1e)

Contract	Location (URL)
AutomatedVaultERC20	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/AutomatedVaultERC20.sol
AutomatedVaultManager	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/AutomatedVaultManager.sol
Bank	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/Bank.sol
Executor	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/executors/Executor.sol
PCSV3Executor01	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/executors/PCSV3Executor01.sol
PCSV3StableExecutor	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/executors/PCSV3StableExecutor.sol
AVManagerV3Gateway	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/gateway/AVManagerV3Gateway.sol
Constants	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/libraries/Constants.sol
LibFixedPoint128	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/libraries/LibFixedPoint128.sol
LibFixedPoint96	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/libraries/LibFixedPoint96.sol
LibFullMath	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d

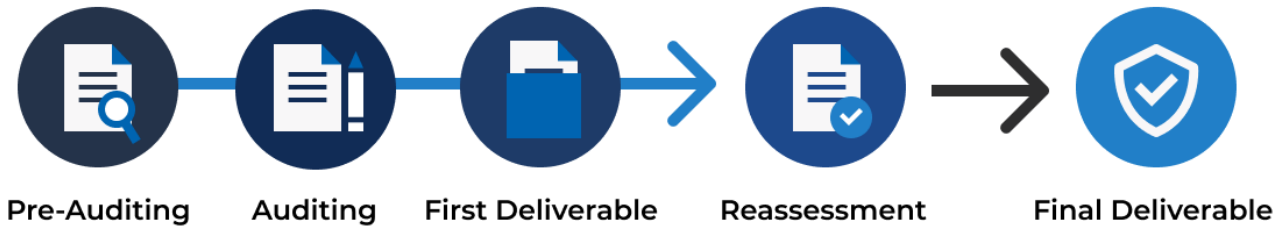
	1464/src/libraries/LibFullMath.sol
LibLiquidityAmounts	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/libraries/LibLiquidityAmounts.sol
LibShareUtil	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/libraries/LibShareUtil.sol
LibSqrtPriceX96	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/libraries/LibSqrtPriceX96.sol
LibTickMath	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/libraries/LibTickMath.sol
BaseOracle	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/oracles/BaseOracle.sol
PancakeV3VaultOracle	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/oracles/PancakeV3VaultOracle.sol
PancakeV3VaultReader	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/reader/PancakeV3VaultReader.sol
PancakeV3Worker	https://github.com/alpaca-finance/alpaca-v2-automated-vault/blob/64711d1464/src/workers/PancakeV3Worker.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit 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 testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/InspexCo/SCSTG/releases/download/v1.0/SCSTG_v1.0.pdf) which covers most prevalent risks in smart contracts. The latest version of the document can also be found at https://inspex.gitbook.io/testing-guide/.

The following audit items were checked during the auditing activity:

Testing Category	Testing Items
1. Architecture and Design	1.1. Proper measures should be used to control the modifications of smart contract logic 1.2. The latest stable compiler version should be used 1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds 1.4. The smart contract source code should be publicly available 1.5. State variables should not be unfairly controlled by privileged accounts 1.6. Least privilege principle should be used for the rights of each role
2. Access Control	2.1. Contract self-destruct should not be done by unauthorized actors 2.2. Contract ownership should not be modifiable by unauthorized actors 2.3. Access control should be defined and enforced for each actor roles 2.4. Authentication measures must be able to correctly identify the user 2.5. Smart contract initialization should be done only once by an authorized party 2.6. tx.origin should not be used for authorization
3. Error Handling and Logging	3.1. Function return values should be checked to handle different results 3.2. Privileged functions or modifications of critical states should be logged 3.3. Modifier should not skip function execution without reverting
4. Business Logic	4.1. The business logic implementation should correspond to the business design 4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions 4.3. msg.value should not be used in loop iteration
5. Blockchain Data	5.1. Result from random value generation should not be predictable 5.2. Spot price should not be used as a data source for price oracles 5.3. Timestamp should not be used to execute critical functions 5.4. Plain sensitive data should not be stored on-chain 5.5. Modification of array state should not be done by value 5.6. State variable should not be used without being initialized

Testing Category	Testing Items
6. External Components	6.1. Unknown external components should not be invoked 6.2. Funds should not be approved or transferred to unknown accounts 6.3. Reentrant calling should not negatively affect the contract states 6.4. Vulnerable or outdated components should not be used in the smart contract 6.5. Deprecated components that have no longer been supported should not be used in the smart contract 6.6. Delegatecall should not be used on untrusted contracts
7. Arithmetic	7.1. Values should be checked before performing arithmetic operations to prevent overflows and underflows 7.2. Explicit conversion of types should be checked to prevent unexpected results 7.3. Integer division should not be done before multiplication to prevent loss of precision
8. Denial of Services	8.1. State changing functions that loop over unbounded data structures should not be used 8.2. Unexpected revert should not make the whole smart contract unusable 8.3. Strict equalities should not cause the function to be unusable
9. Best Practices	9.1. State and function visibility should be explicitly labeled 9.2. Token implementation should comply with the standard specification 9.3. Floating pragma version should not be used 9.4. Builtin symbols should not be shadowed 9.5. Functions that are never called internally should not have public visibility 9.6. Assert statement should not be used for validating common conditions

3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) 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**.

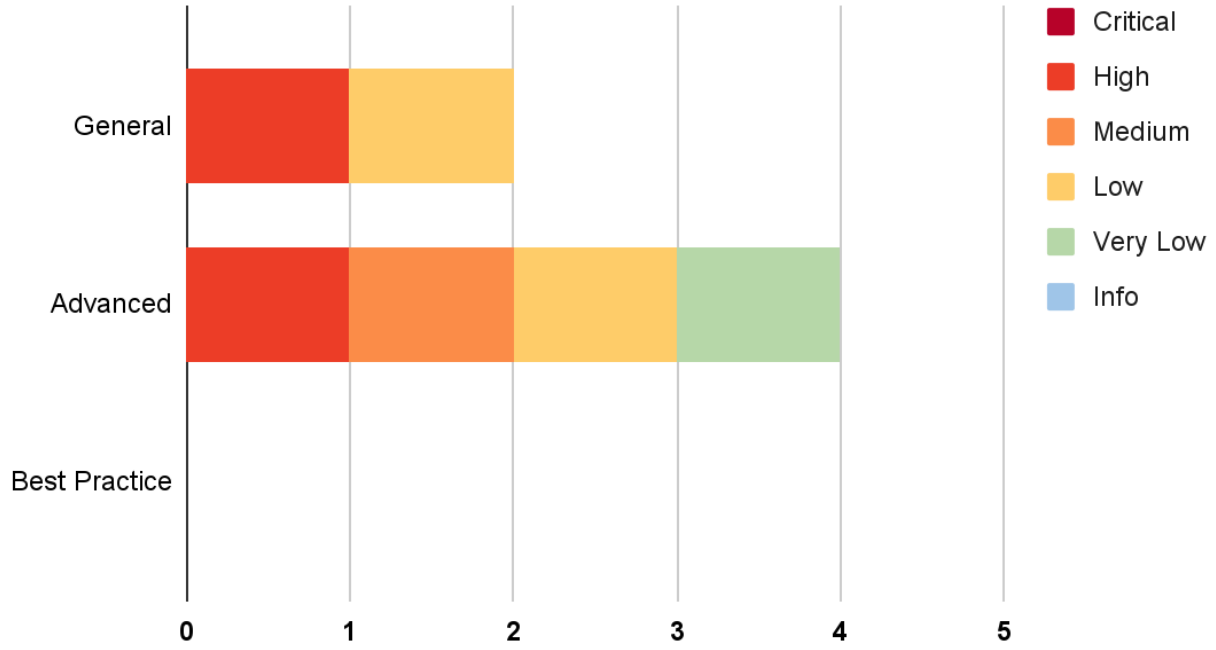
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

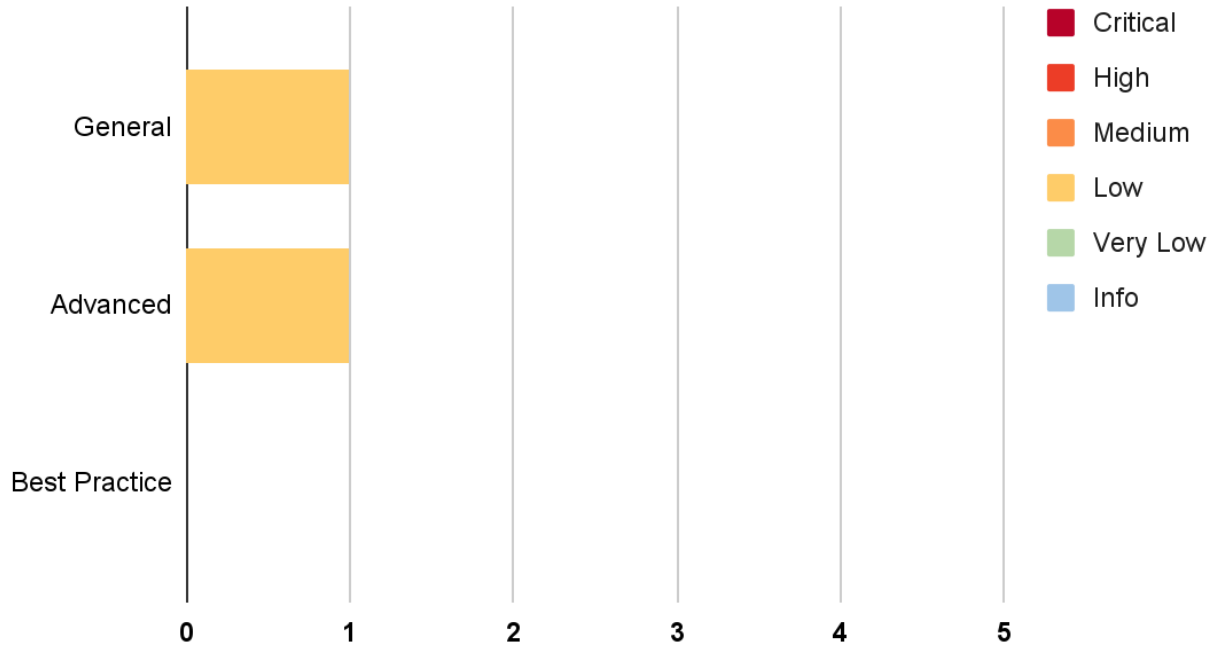
4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

Assessment:



Reassessment:



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	Centralized Control of State Variable	General	High	Resolved *
IDX-002	Denial of Service on Withdrawal	Advanced	High	Resolved
IDX-003	Force Decrease Liquidity on Withdrawal	Advanced	Medium	Resolved *
IDX-004	Transaction Ordering Dependence	General	Low	Acknowledged
IDX-005	PancakeSwap V3 Pair Price Manipulation	Advanced	Low	Acknowledged
IDX-006	Slippage Tolerance Exceed Price Range	Advanced	Very Low	Resolved

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

5. Detailed Findings Information

5.1. Centralized Control of State Variable

ID	IDX-001
Target	AutomatedVaultManager PCSV3Executor01 PCSV3StableExecutor BaseOracle PancakeV3VaultOracle PancakeV3Worker
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood: Medium** There is nothing to restrict the changes from being done; however, this action can only be done by the contract owner.
Status	Resolved * The Alpaca Finance team has mitigated this issue by implementing a Timelock contract as the owner of all contracts to prevent immediate changes. However, the timelock mechanism was not in use at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock mechanism before using the platform.

5.1.1. Description

Critical state variables can be updated at 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, there is currently 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
AutomatedVaultManager (L: 421)	openVault	onlyOwner
AutomatedVaultManager (L: 467)	setVaultTokenImplementation	onlyOwner

AutomatedVaultManager (L: 472)	setManagementFeePerSec	onlyOwner
AutomatedVaultManager (L: 483)	setManagementFeeTreasury	onlyOwner
AutomatedVaultManager (L: 492)	setWithdrawalFeeTreasury	onlyOwner
AutomatedVaultManager (L: 504)	setVaultManager	onlyOwner
AutomatedVaultManager (L: 511)	setAllowToken	onlyOwner
AutomatedVaultManager (L: 519)	setToleranceBps	onlyOwner
AutomatedVaultManager (L: 526)	setMaxLeverage	onlyOwner
AutomatedVaultManager (L: 535)	setMinimumDeposit	onlyOwner
AutomatedVaultManager (L: 546)	setWithdrawalFeeBps	onlyOwner
AutomatedVaultManager (L: 557)	setCapacity	onlyOwner
PCSV3Executor01 (L: 422)	setRepurchaseSlippageBps	onlyOwner
PCSV3Executor01 (L: 430)	setVaultOracle	onlyOwner
PCSV3StableExecutor (L: 421)	setRepurchaseThreshold	onlyOwner
PCSV3StableExecutor (L: 430)	setRepurchaseSlippageBps	onlyOwner
PCSV3StableExecutor (L: 438)	setVaultOracle	onlyOwner
BaseOracle (L: 30)	setPriceFeedOf	onlyOwner
BaseOracle (L: 40)	setMaxPriceAge	onlyOwner
PancakeV3VaultOracle (L: 58)	setMaxPriceDiff	onlyOwner
PancakeV3Worker (L: 587)	setTradingPerformanceFee	onlyOwner
PancakeV3Worker (L: 596)	setRewardPerformanceFee	onlyOwner
PancakeV3Worker (L: 605)	setPerformanceFeeBucket	onlyOwner
PancakeV3Worker (L:613)	setCakeToTokenPath	onlyOwner

5.1.2. 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 implementing a community-run smart contract governance to control the use of these functions.

If removing the functions or implementing the smart contract governance is not possible, Inspex suggests mitigating the risk of this issue by using a timelock mechanism to delay the changes for a reasonable amount of time, at least 24 hours.

Please note that, if the time lock mechanism is applied, the `setIsDepositPaused()` or `setIsWithdrawPaused()` functions will be regularly used by the `onlyOwner` role, we suggest changing the allowed caller of this function to other instead of `onlyOwner` modifier to prevent it from being delayed.

5.2. Denial of Service on Withdrawal

ID	IDX-002
Target	PCSV3Executor01 PCSV3StableExecutor
Category	Advanced Smart Contract Vulnerability
CWE	CWE-703: Improper Check or Handling of Exceptional Conditions
Risk	**Severity: High** **Impact: High** Due to the revert in the decreased liquidity process, the user will be unable to withdraw their funds. **Likelihood: Medium** While users withdraw their funds, the `PancakeSwapV3Worker` contract will decrease liquidity in the PancakeSwap V3 pool. However, the transaction will be reverted if the contract tries to decrease liquidity with a `0` amount.
Status	**Resolved** The Alpaca Finance team fixed this issue by checking the `_liquidity` to not be equal to 0 in `_decreaseLiquidity()` function of `PancakeV3Worker` contract.

5.2.1. Description

In the `PCSV3Executor01` and `PCSV3StableExecutor` contracts, the `onWithdraw()` function is used to withdraw liquidity from the opened PancakeSwap V3 position and the undeployed fund in the `PancakeSwapV3Worker` contract.

In cases where liquidity is provided, this function will reduce the position in a `PancakeSwapV3Worker` contract by executing the `decreasePosition()` function and withdrawing a portion of liquidity based on the number of shares the caller wants to withdraw at line 135 in the `PCSV3StableExecutor` contract. Similarly, at line 146 in the `PCSV3StableExecutor` contract.

PCSV3Executor01.sol

```
112 function onWithdraw(address _worker, address _vaultToken, uint256
    _sharesToWithdraw)
113   external
114   override
115   onlyVaultManager
116   onlyOutOfExecutionScope
117   returns (AutomatedVaultManager.TokenAmount[] memory _results)
118 {
119   uint256 _totalShares = ERC20(_vaultToken).totalSupply();
```

```
120   ERC20 _token0 = PancakeV3Worker(_worker).token0();
121   ERC20 _token1 = PancakeV3Worker(_worker).token1();
122
123   // Withdraw from nft liquidity (if applicable) and undeployed funds
124   uint256 _amount0Withdraw;
125   uint256 _amount1Withdraw;
126   {
127     _amount0Withdraw = _token0.balanceOf(_worker) * _sharesToWithdraw /
_totalShares;
128     _amount1Withdraw = _token1.balanceOf(_worker) * _sharesToWithdraw /
_totalShares;
129     {
130       uint256 _tokenId = PancakeV3Worker(_worker).nftTokenId();
131       if (_tokenId != 0) {
132         (,,,,,,, uint128 _liquidity,,,,) =
PancakeV3Worker(_worker).nftPositionManager().positions(_tokenId);
133         if (_liquidity != 0) {
134           (uint256 _amount0Decreased, uint256 _amount1Decreased) =
135             PancakeV3Worker(_worker).decreasePosition(uint128(_liquidity *
_sharesToWithdraw / _totalShares));
136           // Tokens still with worker after `decreasePosition` so we need to
add to withdrawal
137           _amount0Withdraw += _amount0Decreased;
138           _amount1Withdraw += _amount1Decreased;
139         }
140       }
141     }
142     // Withdraw undeployed funds and decreased liquidity if any
143     if (_amount0Withdraw != 0) {
144       PancakeV3Worker(_worker).transferToExecutor(address(_token0),
_amount0Withdraw);
145     }
146     if (_amount1Withdraw != 0) {
147       PancakeV3Worker(_worker).transferToExecutor(address(_token1),
_amount1Withdraw);
148     }
149   }
150
151   // Repay with amount withdrawn, swap other token to repay token if not enough
152   // NOTE: can't repay if vault has no equity (position value + undeployed
funds < debt value)
153   // due to amount withdrawn is not enough to repay and will revert
154   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
_amount0Withdraw, _token0, _token1);
155   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
_amount1Withdraw, _token1, _token0);
156
```

```
157   // What is left after repayment belongs to user
158   uint256 _amount0AfterRepay = _token0.balanceOf(address(this));
159   if (_amount0AfterRepay != 0) {
160     _token0.safeTransfer(msg.sender, _amount0AfterRepay);
161   }
162   uint256 _amount1AfterRepay = _token1.balanceOf(address(this));
163   if (_amount1AfterRepay != 0) {
164     _token1.safeTransfer(msg.sender, _amount1AfterRepay);
165   }
166
167   emit LogOnWithdraw(
168     _vaultToken,
169     _worker,
170     _sharesToWithdraw,
171     _totalShares,
172     _amount0Withdraw,
173     _amount1Withdraw,
174     _amount0AfterRepay,
175     _amount1AfterRepay
176   );
177
178   _results = new AutomatedVaultManager.TokenAmount[](2);
179   _results[0] = AutomatedVaultManager.TokenAmount({ token: address(_token0),
    amount: _amount0AfterRepay });
180   _results[1] = AutomatedVaultManager.TokenAmount({ token: address(_token1),
    amount: _amount1AfterRepay });
181   return _results;
182 }
```

PCSV3StableExecutor.sol

```
123 function onWithdraw(address _worker, address _vaultToken, uint256
    _sharesToWithdraw)
124   external
125   override
126   onlyVaultManager
127   onlyOutOfExecutionScope
128   returns (AutomatedVaultManager.TokenAmount[] memory _results)
129 {
130   uint256 _totalShares = ERC20(_vaultToken).totalSupply();
131   ERC20 _token0 = PancakeV3Worker(_worker).token0();
132   ERC20 _token1 = PancakeV3Worker(_worker).token1();
133
134   // Withdraw from nft liquidity (if applicable) and undeployed funds
135   uint256 _amount0Withdraw;
136   uint256 _amount1Withdraw;
137   {
138     _amount0Withdraw = _token0.balanceOf(_worker) * _sharesToWithdraw /
```

```
_totalShares;
139     _amount1Withdraw = _token1.balanceOf(_worker) * _sharesToWithdraw /
_totalShares;
140     {
141       uint256 _tokenId = PancakeV3Worker(_worker).nftTokenId();
142       if (_tokenId != 0) {
143         (,,,,,,, uint128 _liquidity,,,,) =
PancakeV3Worker(_worker).nftPositionManager().positions(_tokenId);
144         if (_liquidity != 0) {
145           (uint256 _amount0Decreased, uint256 _amount1Decreased) =
146             PancakeV3Worker(_worker).decreasePosition(uint128(_liquidity *
_sharesToWithdraw / _totalShares));
147           // Tokens still with worker after `decreasePosition` so we need to
add to withdrawal
148           _amount0Withdraw += _amount0Decreased;
149           _amount1Withdraw += _amount1Decreased;
150         }
151       }
152     }
153     // Withdraw undeployed funds and decreased liquidity if any
154     if (_amount0Withdraw != 0) {
155       PancakeV3Worker(_worker).transferToExecutor(address(_token0),
_amount0Withdraw);
156     }
157     if (_amount1Withdraw != 0) {
158       PancakeV3Worker(_worker).transferToExecutor(address(_token1),
_amount1Withdraw);
159     }
160   }
161
162   // Repay with amount withdrawn, swap other token to repay token if not enough
163   // NOTE: can't repay if vault has no equity (position value + undeployed
funds < debt value)
164   // due to amount withdrawn is not enough to repay and will revert
165   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
_amount0Withdraw, _token0, _token1);
166   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
_amount1Withdraw, _token1, _token0);
167
168   // What is left after repayment belongs to user
169   uint256 _amount0AfterRepay = _token0.balanceOf(address(this));
170   if (_amount0AfterRepay != 0) {
171     _token0.safeTransfer(msg.sender, _amount0AfterRepay);
172   }
173   uint256 _amount1AfterRepay = _token1.balanceOf(address(this));
174   if (_amount1AfterRepay != 0) {
175     _token1.safeTransfer(msg.sender, _amount1AfterRepay);
```

```
176   }
177
178   emit LogOnWithdraw(
179     _vaultToken,
180     _worker,
181     _sharesToWithdraw,
182     _totalShares,
183     _amount0Withdraw,
184     _amount1Withdraw,
185     _amount0AfterRepay,
186     _amount1AfterRepay
187   );
188
189   _results = new AutomatedVaultManager.TokenAmount[](2);
190   _results[0] = AutomatedVaultManager.TokenAmount({ token: address(_token0),
      amount: _amount0AfterRepay });
191   _results[1] = AutomatedVaultManager.TokenAmount({ token: address(_token1),
      amount: _amount1AfterRepay });
192   return _results;
193 }
```

Since the amount of decreased liquidity depends on `_liquidity` multiplied by `_sharesToWithdraw` divided by `_totalShares`, the user may decrease the liquidity position by `0` when there is very low liquidity or a very small portion of the user's shares at line 433 and 444 in the `PancakeSwapV3Worker` contract.

PancakeSwapV3Worker.sol

```
423 function decreasePosition(uint128 _liquidity)
424   external
425   nonReentrant
426   onlyExecutorInScope
427   returns (uint256 _amount0, uint256 _amount1)
428 {
429   uint256 _nftTokenId = nftTokenId;
430   if (_nftTokenId == 0) {
431     revert PancakeV3Worker_PositionNotExist();
432   }
433   (_amount0, _amount1) = _decreaseLiquidity(_nftTokenId, masterChef,
      _liquidity);
434   emit LogDecreasePosition(_nftTokenId, msg.sender, _amount0, _amount1,
      _liquidity);
435 }
436
437 function _decreaseLiquidity(uint256 _nftTokenId, IPancakeV3MasterChef
      _masterChef, uint128 _liquidity)
438   internal
439   returns (uint256 _amount0, uint256 _amount1)
```

```
440 {
441   // claim all rewards accrued before removing liquidity from LP
442   _harvest();
443
444   _masterChef.decreaseLiquidity(
445     IPancakeV3MasterChef.DecreaseLiquidityParams({
446       tokenId: _nftTokenId,
447       liquidity: _liquidity,
448       amount0Min: 0,
449       amount1Min: 0,
450       deadline: block.timestamp
451     })
452   );
453   (_amount0, _amount1) = _masterChef.collect(
454     IPancakeV3MasterChef.CollectParams({
455       tokenId: _nftTokenId,
456       recipient: address(this),
457       amount0Max: type(uint128).max,
458       amount1Max: type(uint128).max
459     })
460   );
461 }
```

The PancakeSwap V3's `_masterChef.decreaseLiquidity()` function calls the `NonfungiblePositionManager.decreaseLiquidity()` function, which requires the `params.liquidity` (amount of liquidity to decrease) to be greater than `0` at line 266. If the `PancakeSwapV3Worker` attempts to decrease the liquidity with a `0` amount, the withdrawal transaction will revert.

NonfungiblePositionManager.sol

```
258 function decreaseLiquidity(DecreaseLiquidityParams calldata params)
259     external
260     payable
261     override
262     isAuthorizedForToken(params.tokenId)
263     checkDeadline(params.deadline)
264     returns (uint256 amount0, uint256 amount1)
265 {
266     require(params.liquidity > 0);
267     Position storage position = _positions[params.tokenId];
268
269     uint128 positionLiquidity = position.liquidity;
270     require(positionLiquidity >= params.liquidity);
271
272     PoolAddress.PoolKey memory poolKey = _poolIdToPoolKey[position.poolId];
273     IPancakeV3Pool pool = IPancakeV3Pool(PoolAddress.computeAddress(deployer,
poolKey));
```

```
274     (amount0, amount1) = pool.burn(position.tickLower, position.tickUpper,
    params.liquidity);
275
276     require(amount0 >= params.amount0Min && amount1 >= params.amount1Min,
    'Price slippage check');
277
278     bytes32 positionKey = PositionKey.compute(address(this),
    position.tickLower, position.tickUpper);
279     // this is now updated to the current transaction
280     (, uint256 feeGrowthInside0LastX128, uint256 feeGrowthInside1LastX128, , )
    = pool.positions(positionKey);
281
282     position.tokensOwed0 +=
283         uint128(amount0) +
284         uint128(
285             FullMath.mulDiv(
286                 feeGrowthInside0LastX128 - position.feeGrowthInside0LastX128,
287                 positionLiquidity,
288                 FixedPoint128.Q128
289             )
290         );
291     position.tokensOwed1 +=
292         uint128(amount1) +
293         uint128(
294             FullMath.mulDiv(
295                 feeGrowthInside1LastX128 - position.feeGrowthInside1LastX128,
296                 positionLiquidity,
297                 FixedPoint128.Q128
298             )
299         );
300
301     position.feeGrowthInside0LastX128 = feeGrowthInside0LastX128;
302     position.feeGrowthInside1LastX128 = feeGrowthInside1LastX128;
303     // subtraction is safe because we checked positionLiquidity is gte
    params.liquidity
304     position.liquidity = positionLiquidity - params.liquidity;
305
306     emit DecreaseLiquidity(params.tokenId, params.liquidity, amount0, amount1);
307 }
```

As a result, users will be unable to withdraw funds from the platform.

5.2.2. Remediation

Inspex suggests adding the condition to check whether the `(_liquidity * _sharesToWithdraw / _totalShares)` value is not equal to `0` as shown in lines 133 of `PCSV3Executor01.sol` and 144 of `PCSV3StableExecutor.sol` to skip reverting from the decrease position with the `0` amount.

PCSV3Executor01.sol

```
112 function onWithdraw(address _worker, address _vaultToken, uint256
    _sharesToWithdraw)
113   external
114   override
115   onlyVaultManager
115   onlyOutOfExecutionScope
117   returns (AutomatedVaultManager.TokenAmount[] memory _results)
118 {
119   uint256 _totalShares = ERC20(_vaultToken).totalSupply();
120   ERC20 _token0 = PancakeV3Worker(_worker).token0();
121   ERC20 _token1 = PancakeV3Worker(_worker).token1();
122
123   // Withdraw from nft liquidity (if applicable) and undeployed funds
124   uint256 _amount0Withdraw;
125   uint256 _amount1Withdraw;
126   {
127     _amount0Withdraw = _token0.balanceOf(_worker) * _sharesToWithdraw /
    _totalShares;
128     _amount1Withdraw = _token1.balanceOf(_worker) * _sharesToWithdraw /
    _totalShares;
129     {
130       uint256 _tokenId = PancakeV3Worker(_worker).nftTokenId();
131       if (_tokenId != 0) {
132         (,,,,,,, uint128 _liquidity,,,,) =
    PancakeV3Worker(_worker).nftPositionManager().positions(_tokenId);
133         if ((_liquidity * _sharesToWithdraw / _totalShares) != 0) {
134           (uint256 _amount0Decreased, uint256 _amount1Decreased) =
135             PancakeV3Worker(_worker).decreasePosition(uint128(_liquidity *
    _sharesToWithdraw / _totalShares));
136           // Tokens still with worker after `decreasePosition` so we need to
    add to withdrawal
137           _amount0Withdraw += _amount0Decreased;
138           _amount1Withdraw += _amount1Decreased;
139         }
140       }
141     }
142     // Withdraw undeployed funds and decreased liquidity if any
143     if (_amount0Withdraw != 0) {
144       PancakeV3Worker(_worker).transferToExecutor(address(_token0),
    _amount0Withdraw);
145     }
146     if (_amount1Withdraw != 0) {
147       PancakeV3Worker(_worker).transferToExecutor(address(_token1),
    _amount1Withdraw);
148     }
149   }
```

```
150
151   // Repay with amount withdrawn, swap other token to repay token if not enough
152   // NOTE: can't repay if vault has no equity (position value + undeployed
      funds < debt value)
153   // due to amount withdrawn is not enough to repay and will revert
154   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
      _amount0Withdraw, _token0, _token1);
155   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
      _amount1Withdraw, _token1, _token0);
156
157   // What is left after repayment belongs to user
158   uint256 _amount0AfterRepay = _token0.balanceOf(address(this));
159   if (_amount0AfterRepay != 0) {
160     _token0.safeTransfer(msg.sender, _amount0AfterRepay);
161   }
162   uint256 _amount1AfterRepay = _token1.balanceOf(address(this));
163   if (_amount1AfterRepay != 0) {
164     _token1.safeTransfer(msg.sender, _amount1AfterRepay);
165   }
166
167   emit LogOnWithdraw(
168     _vaultToken,
169     _worker,
170     _sharesToWithdraw,
171     _totalShares,
172     _amount0Withdraw,
173     _amount1Withdraw,
174     _amount0AfterRepay,
175     _amount1AfterRepay
176   );
177
178   _results = new AutomatedVaultManager.TokenAmount[](2);
179   _results[0] = AutomatedVaultManager.TokenAmount({ token: address(_token0),
      amount: _amount0AfterRepay });
180   _results[1] = AutomatedVaultManager.TokenAmount({ token: address(_token1),
      amount: _amount1AfterRepay });
181   return _results;
182 }
```

PCSV3StableExecutor.sol

```
123 function onWithdraw(address _worker, address _vaultToken, uint256
    _sharesToWithdraw)
124   external
125   override
126   onlyVaultManager
127   onlyOutOfExecutionScope
128   returns (AutomatedVaultManager.TokenAmount[] memory _results)
```

```
129 {
130   uint256 _totalShares = ERC20(_vaultToken).totalSupply();
131   ERC20 _token0 = PancakeV3Worker(_worker).token0();
132   ERC20 _token1 = PancakeV3Worker(_worker).token1();
133 
134   // Withdraw from nft liquidity (if applicable) and undeployed funds
135   uint256 _amount0Withdraw;
136   uint256 _amount1Withdraw;
137   {
138     _amount0Withdraw = _token0.balanceOf(_worker) * _sharesToWithdraw /
_totalShares;
139     _amount1Withdraw = _token1.balanceOf(_worker) * _sharesToWithdraw /
_totalShares;
140     {
141       uint256 _tokenId = PancakeV3Worker(_worker).nftTokenId();
142       if (_tokenId != 0) {
143         (,,,,,,, uint128 _liquidity,,,,) =
PancakeV3Worker(_worker).nftPositionManager().positions(_tokenId);
144         if ((_liquidity * _sharesToWithdraw / _totalShares) != 0) {
145           (uint256 _amount0Decreased, uint256 _amount1Decreased) =
146             PancakeV3Worker(_worker).decreasePosition(uint128(_liquidity *
_sharesToWithdraw / _totalShares));
147           // Tokens still with worker after `decreasePosition` so we need to
add to withdrawal
148           _amount0Withdraw += _amount0Decreased;
149           _amount1Withdraw += _amount1Decreased;
150         }
151       }
152     }
153     // Withdraw undeployed funds and decreased liquidity if any
154     if (_amount0Withdraw != 0) {
155       PancakeV3Worker(_worker).transferToExecutor(address(_token0),
_amount0Withdraw);
156     }
157     if (_amount1Withdraw != 0) {
158       PancakeV3Worker(_worker).transferToExecutor(address(_token1),
_amount1Withdraw);
159     }
160   }
161 
162   // Repay with amount withdrawn, swap other token to repay token if not enough
163   // NOTE: can't repay if vault has no equity (position value + undeployed
funds < debt value)
164   // due to amount withdrawn is not enough to repay and will revert
165   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
_amount0Withdraw, _token0, _token1);
166   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
```

```
    _amount1Withdraw, _token1, _token0);
167
168   // What is left after repayment belongs to user
169   uint256 _amount0AfterRepay = _token0.balanceOf(address(this));
170   if (_amount0AfterRepay != 0) {
171     _token0.safeTransfer(msg.sender, _amount0AfterRepay);
172   }
173   uint256 _amount1AfterRepay = _token1.balanceOf(address(this));
174   if (_amount1AfterRepay != 0) {
175     _token1.safeTransfer(msg.sender, _amount1AfterRepay);
176   }
177
178   emit LogOnWithdraw(
179     _vaultToken,
180     _worker,
181     _sharesToWithdraw,
182     _totalShares,
183     _amount0Withdraw,
184     _amount1Withdraw,
185     _amount0AfterRepay,
186     _amount1AfterRepay
187   );
188
189   _results = new AutomatedVaultManager.TokenAmount[](2);
190   _results[0] = AutomatedVaultManager.TokenAmount({ token: address(_token0),
    amount: _amount0AfterRepay });
191   _results[1] = AutomatedVaultManager.TokenAmount({ token: address(_token1),
    amount: _amount1AfterRepay });
192   return _results;
193 }
```

5.3. Force Decrease Liquidity on Withdrawal

ID	IDX-003
Target	PCSV3Executor01 PCSV3StableExecutor
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The liquidity provided to PancakeSwap V3 can be forcefully decreased, resulting in platform users receiving less reward. **Likelihood:** Medium This issue occurs every time that the `onWithdraw()` function is called during the withdrawal process. However, the decreased amount will depend on the portion of the withdrawal shares.
Status	Resolved * To mitigate the issue, the Alpaca Finance team clarified that the logic was intentional to keep every composition of the vault intact after withdrawal. This implies that the vault just scales down proportionately. Meaning the share owner is proportionately and equally owner every aspect of the vault e.g. position, undeployed funds, debt. To mitigate this attack on thinning the farmed position, the Alpaca Finance team has applied a withdrawal fee on the platform. Repetitive deposits and withdrawals are economically discouraged to do so as the manager will collect the withdrawal fee and redeploy the fund again.

5.3.1. Description

Every time the user withdraws funds, the `onWithdraw()` function will always be executed. If PancakeSwap V3's position is opened, this function will withdraw a portion of liquidity by decreasing the position liquidity at line 135 in the `PCSV3Executor01` contract and at line 146 in the `PCSV3StableExecutor` contract.

PCSV3Executor01.sol

```
112 function onWithdraw(address _worker, address _vaultToken, uint256
    _sharesToWithdraw)
113   external
114   override
115   onlyVaultManager
116   onlyOutOfExecutionScope
```

```
117   returns (AutomatedVaultManager.TokenAmount[] memory _results)
118 {
119   uint256 _totalShares = ERC20(_vaultToken).totalSupply();
120   ERC20 _token0 = PancakeV3Worker(_worker).token0();
121   ERC20 _token1 = PancakeV3Worker(_worker).token1();
122
123   // Withdraw from nft liquidity (if applicable) and undeployed funds
124   uint256 _amount0Withdraw;
125   uint256 _amount1Withdraw;
126   {
127     _amount0Withdraw = _token0.balanceOf(_worker) * _sharesToWithdraw /
_totalShares;
128     _amount1Withdraw = _token1.balanceOf(_worker) * _sharesToWithdraw /
_totalShares;
129     {
130       uint256 _tokenId = PancakeV3Worker(_worker).nftTokenId();
131       if (_tokenId != 0) {
132         (,,,,,,, uint128 _liquidity,,,,) =
PancakeV3Worker(_worker).nftPositionManager().positions(_tokenId);
133         if (_liquidity != 0) {
134           (uint256 _amount0Decreased, uint256 _amount1Decreased) =
135             PancakeV3Worker(_worker).decreasePosition(uint128(_liquidity *
_sharesToWithdraw / _totalShares));
136           // Tokens still with worker after `decreasePosition` so we need to
add to withdrawal
137           _amount0Withdraw += _amount0Decreased;
138           _amount1Withdraw += _amount1Decreased;
139         }
140       }
141     }
142     // Withdraw undeployed funds and decreased liquidity if any
143     if (_amount0Withdraw != 0) {
144       PancakeV3Worker(_worker).transferToExecutor(address(_token0),
_amount0Withdraw);
145     }
146     if (_amount1Withdraw != 0) {
147       PancakeV3Worker(_worker).transferToExecutor(address(_token1),
_amount1Withdraw);
148     }
149   }
150
151   // Repay with amount withdrawn, swap other token to repay token if not enough
152   // NOTE: can't repay if vault has no equity (position value + undeployed
funds < debt value)
153   // due to amount withdrawn is not enough to repay and will revert
154   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
_amount0Withdraw, _token0, _token1);
```

```
155   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
      _amount1Withdraw, _token1, _token0);
156
157   // What is left after repayment belongs to user
158   uint256 _amount0AfterRepay = _token0.balanceOf(address(this));
159   if (_amount0AfterRepay != 0) {
160     _token0.safeTransfer(msg.sender, _amount0AfterRepay);
161   }
162   uint256 _amount1AfterRepay = _token1.balanceOf(address(this));
163   if (_amount1AfterRepay != 0) {
164     _token1.safeTransfer(msg.sender, _amount1AfterRepay);
165   }
166
167   emit LogOnWithdraw(
168     _vaultToken,
169     _worker,
170     _sharesToWithdraw,
171     _totalShares,
172     _amount0Withdraw,
173     _amount1Withdraw,
174     _amount0AfterRepay,
175     _amount1AfterRepay
176   );
177
178   _results = new AutomatedVaultManager.TokenAmount[](2);
179   _results[0] = AutomatedVaultManager.TokenAmount({ token: address(_token0),
      amount: _amount0AfterRepay });
180   _results[1] = AutomatedVaultManager.TokenAmount({ token: address(_token1),
      amount: _amount1AfterRepay });
181   return _results;
182 }
```

PCSV3StableExecutor.sol

```
123 function onWithdraw(address _worker, address _vaultToken, uint256
    _sharesToWithdraw)
124   external
125   override
126   onlyVaultManager
127   onlyOutOfExecutionScope
128   returns (AutomatedVaultManager.TokenAmount[] memory _results)
129 {
130   uint256 _totalShares = ERC20(_vaultToken).totalSupply();
131   ERC20 _token0 = PancakeV3Worker(_worker).token0();
132   ERC20 _token1 = PancakeV3Worker(_worker).token1();
133
134   // Withdraw from nft liquidity (if applicable) and undeployed funds
135   uint256 _amount0Withdraw;
```

```
136   uint256 _amount1Withdraw;
137   {
138     _amount0Withdraw = _token0.balanceOf(_worker) * _sharesToWithdraw /
      _totalShares;
139     _amount1Withdraw = _token1.balanceOf(_worker) * _sharesToWithdraw /
      _totalShares;
140     {
141       uint256 _tokenId = PancakeV3Worker(_worker).nftTokenId();
142       if (_tokenId != 0) {
143         (,,,,,,, uint128 _liquidity,,,,) =
      PancakeV3Worker(_worker).nftPositionManager().positions(_tokenId);
144         if (_liquidity != 0) {
145           (uint256 _amount0Decreased, uint256 _amount1Decreased) =
146             PancakeV3Worker(_worker).decreasePosition(uint128(_liquidity *
      _sharesToWithdraw / _totalShares));
147           // Tokens still with worker after `decreasePosition` so we need to
      add to withdrawal
148           _amount0Withdraw += _amount0Decreased;
149           _amount1Withdraw += _amount1Decreased;
150         }
151       }
152     }
153     // Withdraw undeployed funds and decreased liquidity if any
154     if (_amount0Withdraw != 0) {
155       PancakeV3Worker(_worker).transferToExecutor(address(_token0),
      _amount0Withdraw);
156     }
157     if (_amount1Withdraw != 0) {
158       PancakeV3Worker(_worker).transferToExecutor(address(_token1),
      _amount1Withdraw);
159     }
160   }
161
162   // Repay with amount withdrawn, swap other token to repay token if not enough
163   // NOTE: can't repay if vault has no equity (position value + undeployed
      funds < debt value)
164   // due to amount withdrawn is not enough to repay and will revert
165   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
      _amount0Withdraw, _token0, _token1);
166   _repayOnWithdraw(_worker, _vaultToken, _sharesToWithdraw, _totalShares,
      _amount1Withdraw, _token1, _token0);
167
168   // What is left after repayment belongs to user
169   uint256 _amount0AfterRepay = _token0.balanceOf(address(this));
170   if (_amount0AfterRepay != 0) {
171     _token0.safeTransfer(msg.sender, _amount0AfterRepay);
172   }
```

```
173   uint256 _amount1AfterRepay = _token1.balanceOf(address(this));
174   if (_amount1AfterRepay != 0) {
175     _token1.safeTransfer(msg.sender, _amount1AfterRepay);
176   }
177
178   emit LogOnWithdraw(
179     _vaultToken,
180     _worker,
181     _sharesToWithdraw,
182     _totalShares,
183     _amount0Withdraw,
184     _amount1Withdraw,
185     _amount0AfterRepay,
186     _amount1AfterRepay
187   );
188
189   _results = new AutomatedVaultManager.TokenAmount[](2);
190   _results[0] = AutomatedVaultManager.TokenAmount({ token: address(_token0),
    amount: _amount0AfterRepay });
191   _results[1] = AutomatedVaultManager.TokenAmount({ token: address(_token1),
    amount: _amount1AfterRepay });
192   return _results;
193 }
```

The attacker could deposit and then withdraw immediately to forcibly decrease the liquidity. As a result, the liquidity provided to PancakeSwap V3 will be lower than expected, leading to lesser rewards received.

5.3.2. Remediation

Inspex suggests modifying the `onWithdraw()` function to first withdraw funds from the `PancakeSwapV3Worker` contract. If the withdrawn funds are insufficient, it will then proceed to withdraw additional funds from PancakeSwap V3's liquidity position.

5.4. Transaction Ordering Dependence

ID	IDX-004
Target	PancakeV3Worker
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 loss of user reward. **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 clarified that the trade off between gas usage for slippage computation and the impact is not in favor of doing so. Since the UniV3 like LP requires frequent active management, they need the gas to be as efficient as possible. Moreover, the harvest function will be periodically called to mitigate the impact of front running attack

5.4.1. Description

When slippage protection is not in place, it allows attackers to perform a front-running attack by wrapping unprotected transactions with pump and dump, resulting in a bad swapping rate.

The following table contains a list of functions for which slippage protection is not implemented.

Target	Contract	Function	PCSV3 Function
PancakeV3Worker.sol (L: 191)	PancakeV3Worker	openPosition()	mint()
PancakeV3Worker.sol (L: 246)	PancakeV3Worker	increasePosition()	increaseLiquidity()
PancakeV3Worker.sol (L: 555)	PancakeV3Worker	_harvest()	exactInput()

For example, in the harvest process the `_harvest()` function calls the `exactInput()` function without the `amountOutMinimum` parameter specified in lines 555-562.

PancakeV3Worker.sol

```
482 function _harvest() internal {
483   // Skip harvest if already done before in same block
484   if (block.timestamp == lastHarvest) return;
485   lastHarvest = uint40(block.timestamp);
486 
487   uint256 _nftTokenId = nftTokenId;
488   // If tokenId is 0, then nothing to harvest
489   if (_nftTokenId == 0) return;
490 
491   HarvestFeeLocalVars memory _vars;
492 
493   // SLOADs
494   address _performanceFeeBucket = performanceFeeBucket;
495   ERC20 _token0 = token0;
496   ERC20 _token1 = token1;
497   ERC20 _cake = cake;
498   IPancakeV3MasterChef _masterChef = masterChef;
499 
500   // Handle trading fee
501   (_vars.fee0, _vars.fee1) = _masterChef.collect(
502     IPancakeV3MasterChef.CollectParams({
503       tokenId: _nftTokenId,
504       recipient: address(this),
505       amount0Max: type(uint128).max,
506       amount1Max: type(uint128).max
507     })
508   );
509   // Collect performance fee on collected trading fee
510   _vars.tradingPerformanceFeeBps = tradingPerformanceFeeBps;
511   if (_vars.fee0 > 0) {
512     // Safe to unchecked because fee always less than MAX_BPS
513     unchecked {
514       _token0.safeTransfer(_performanceFeeBucket, _vars.fee0 *
_vars.tradingPerformanceFeeBps / MAX_BPS);
515     }
516   }
517   if (_vars.fee1 > 0) {
518     // Safe to unchecked because fee always less than MAX_BPS
519     unchecked {
520       _token1.safeTransfer(_performanceFeeBucket, _vars.fee1 *
_vars.tradingPerformanceFeeBps / MAX_BPS);
521     }
522   }
523 
524   // Handle CAKE rewards
525   _vars.cakeRewards = _masterChef.harvest(_nftTokenId, address(this));
526   if (_vars.cakeRewards > 0) {
```

```
527     uint256 _cakePerformanceFee;
528     // Collect CAKE performance fee
529     // Safe to unchecked because fee always less than MAX_BPS
530     unchecked {
531       _vars.rewardPerformanceFeeBps = rewardPerformanceFeeBps;
532       _cakePerformanceFee = _vars.cakeRewards * _vars.rewardPerformanceFeeBps /
MAX_BPS;
533       _cake.safeTransfer(_performanceFeeBucket, _cakePerformanceFee);
534     }
535
536     // Sell CAKE for token0 or token1, if any
537     // Find out need to sell CAKE to which side by checking currTick
538     (, int24 _currTick,,,,,) = pool.slot0();
539     address _tokenOut = address(_token0);
540     if (_currTick - posTickLower > posTickUpper - _currTick) {
541       // If currTick is closer to tickUpper, then we will sell CAKE for token1
542       _tokenOut = address(_token1);
543     }
544
545     if (_tokenOut != address(_cake)) {
546       IPancakeV3Router _router = router;
547       // Swap reward after fee to token0 or token1
548       // Safe to unchecked because _cakePerformanceFee is always less than
_vars.cakeRewards (see above)
549       uint256 _swapAmount;
550       unchecked {
551         _swapAmount = _vars.cakeRewards - _cakePerformanceFee;
552       }
553       _cake.safeApprove(address(_router), _swapAmount);
554       // Swap CAKE for token0 or token1 based on predefined v3 path
555       _router.exactInput(
556         IPancakeV3Router.ExactInputParams({
557           path: cakeToTokenPath[_tokenOut],
558           recipient: address(this),
559           amountIn: _swapAmount,
560           amountOutMinimum: 0
561         })
562       );
563     }
564   }
565
566   emit LogHarvest(
567     _vars.fee0, _vars.fee1, _vars.tradingPerformanceFeeBps, _vars.cakeRewards,
_vars.rewardPerformanceFeeBps
568   );
569 }
```

5.4.2. 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, and setting it to the `amountOutMin` parameter while calling the `exactInput()` function.

5.5. PancakeSwap V3 Pair Price Manipulation

ID	IDX-005
Target	PCSV3StableExecutor PancakeV3VaultOracle PancakeV3Worker
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Medium There are several uses of returns from the `PancakeV3Pool.slot0()` function that can be easily manipulated, resulting in unexpected behavior on the platform. **Likelihood:** Low These functions are likely to be attacked in order to manipulate the result; however, this requires a large amount of funds with low or no rewards.
Status	**Acknowledged** The Alpaca Finance team acknowledged this issue due to its low to no impact on the platform.

5.5.1. Description

The `PancakeV3Pool.slot0()` function returns several current pool parameters; however, slot0 is not designed to be used as a price oracle because there is no price manipulation prevention mechanism.

The following table contain the list of the `PancakeV3Pool.slot0()` function usage functions:

Target	Contract	Function
PCSV3StableExecutor.sol (L: 334, 341)	PCSV3StableExecutor	repurchase()
PancakeV3VaultOracle.sol (L: 200)	PancakeV3VaultOracle	getExposure()
PancakeV3Worker.sol (L: 272)	PancakeV3Worker	_prepareOptimalTokensForIncrease()
PancakeV3Worker.sol (L: 538)	PancakeV3Worker	_harvest()

For example, in the `_harvest()` function, if the `PancakeV3Pool.slot0()` returns have been manipulated, the `_currTick` value would be changed, resulting in output tokens swapping from $CAKE also being changed.

PancakeV3Worker.sol

```
482 function _harvest() internal {
483   // Skip harvest if already done before in same block
484   if (block.timestamp == lastHarvest) return;
485   lastHarvest = uint40(block.timestamp);
486
487   uint256 _nftTokenId = nftTokenId;
488   // If tokenId is 0, then nothing to harvest
489   if (_nftTokenId == 0) return;
490
491   HarvestFeeLocalVars memory _vars;
492
493   // SLOADs
494   address _performanceFeeBucket = performanceFeeBucket;
495   ERC20 _token0 = token0;
496   ERC20 _token1 = token1;
497   ERC20 _cake = cake;
498   IPancakeV3MasterChef _masterChef = masterChef;
499
500   // Handle trading fee
501   (_vars.fee0, _vars.fee1) = _masterChef.collect(
502     IPancakeV3MasterChef.CollectParams({
503       tokenId: _nftTokenId,
504       recipient: address(this),
505       amount0Max: type(uint128).max,
506       amount1Max: type(uint128).max
507     })
508   );
509   // Collect performance fee on collected trading fee
510   _vars.tradingPerformanceFeeBps = tradingPerformanceFeeBps;
511   if (_vars.fee0 > 0) {
512     // Safe to unchecked because fee always less than MAX_BPS
513     unchecked {
514       _token0.safeTransfer(_performanceFeeBucket, _vars.fee0 *
    _vars.tradingPerformanceFeeBps / MAX_BPS);
515     }
516   }
517   if (_vars.fee1 > 0) {
518     // Safe to unchecked because fee always less than MAX_BPS
519     unchecked {
520       _token1.safeTransfer(_performanceFeeBucket, _vars.fee1 *
    _vars.tradingPerformanceFeeBps / MAX_BPS);
521     }
522   }
523
524   // Handle CAKE rewards
525   _vars.cakeRewards = _masterChef.harvest(_nftTokenId, address(this));
526   if (_vars.cakeRewards > 0) {
```

```
527     uint256 _cakePerformanceFee;
528     // Collect CAKE performance fee
529     // Safe to unchecked because fee always less than MAX_BPS
530     unchecked {
531       _vars.rewardPerformanceFeeBps = rewardPerformanceFeeBps;
532       _cakePerformanceFee = _vars.cakeRewards * _vars.rewardPerformanceFeeBps /
    MAX_BPS;
533       _cake.safeTransfer(_performanceFeeBucket, _cakePerformanceFee);
534     }
535
536     // Sell CAKE for token0 or token1, if any
537     // Find out need to sell CAKE to which side by checking currTick
538     (, int24 _currTick,,,,,) = pool.slot0();
539     address _tokenOut = address(_token0);
540     if (_currTick - posTickLower > posTickUpper - _currTick) {
541       // If currTick is closer to tickUpper, then we will sell CAKE for token1
542       _tokenOut = address(_token1);
543     }
544
545     if (_tokenOut != address(_cake)) {
546       IPancakeV3Router _router = router;
547       // Swap reward after fee to token0 or token1
548       // Safe to unchecked because _cakePerformanceFee is always less than
    _vars.cakeRewards (see above)
549       uint256 _swapAmount;
550       unchecked {
551         _swapAmount = _vars.cakeRewards - _cakePerformanceFee;
552       }
553       _cake.safeApprove(address(_router), _swapAmount);
554       // Swap CAKE for token0 or token1 based on predefined v3 path
555       _router.exactInput(
556         IPancakeV3Router.ExactInputParams({
557           path: cakeToTokenPath[_tokenOut],
558           recipient: address(this),
559           amountIn: _swapAmount,
560           amountOutMinimum: 0
561         })
562       );
563     }
564   }
565
566   emit LogHarvest(
567     _vars.fee0, _vars.fee1, _vars.tradingPerformanceFeeBps, _vars.cakeRewards,
    _vars.rewardPerformanceFeeBps
568   );
569 }
```

5.5.2. Remediation

Inspex suggests using the TWAP oracle instead to prevent the price manipulation attack. For example, use the `consult()` function in the `OracleLibrary` contract (https://github.com/Uniswap/v3-periphery/blob/main/contracts/libraries/OracleLibrary.sol) to get the mean tick from a specific time period.

PancakeV3Worker.sol

```
536     // Sell CAKE for token0 or token1, if any
537     // Find out need to sell CAKE to which side by checking currTick
538     (int24 _currTick,) = OracleLibrary.consult(address(pool), TWAP_PERIOD);
539     address _tokenOut = address(_token0);
540     if (_currTick - posTickLower > posTickUpper - _currTick) {
541       // If currTick is closer to tickUpper, then we will sell CAKE for token1
542       _tokenOut = address(_token1);
543     }
```

Further information about the TWAP oracle usage can be found at: https://uniswapv3book.com/docs/milestone_5/price-oracle/#reading-observations.

5.6. Slippage Tolerance Exceed Price Range

ID	IDX-006
Target	PancakeV3Worker
Category	Advanced Smart Contract Vulnerability
CWE	CWE-703: Improper Check or Handling of Exceptional Conditions
Risk	**Severity:** Very Low **Impact:** Low The `increasePosition()` function call may be reverted; a large volume can shift the price across the position range, resulting in the amount to add liquidity being invalid and could lead to the operation's failure. **Likelihood:** Low It is unlikely to occur because it requires a position with a narrow price range and a large volume to increase the position.
Status	Resolved The Alpaca Finance team fixed this by configuring the swapping price limit to align with the position range's upper or lower boundary.

5.6.1. Description

In the `PancakeV3Worker` contract, there exists a mechanism designed to adjust the current tick of the pool, bringing it back inside the platform's position range when the current tick is outside that range in the increased position process. This is achieved by swapping an asset within the `PancakeV3Worker` contract using the `_prepareOptimalTokensForIncreaseOutOfRange()` function.

PancakeV3Worker.sol

```
337 function _prepareOptimalTokensForIncreaseOutOfRange(
338   address _token0,
339   address _token1,
340   int24 _currTick,
341   int24 _tickLower,
342   int24 _tickUpper,
343   uint256 _amountIn0,
344   uint256 _amountIn1
345 ) internal returns (uint256 _optimalAmount0, uint256 _optimalAmount1) {
346   // SLOAD
347   int24 _tickSpacing = pool.tickSpacing();
348
349   // If out of upper range (currTick > tickUpper), we swap token0 for token1
350   // and vice versa, to push price closer to range.
```

```
351   // We only want to swap until price move back in range so
352   // we will swap until price hit the first tick within range.
353   if (_currTick > _tickUpper) {
354     if (_amountIn0 > 0) {
355       uint256 _token0Before = ERC20(_token0).balanceOf(address(this));
356       // zero for one swap
357       ERC20(_token0).safeApprove(address(router), _amountIn0);
358       uint256 _amountOut = router.exactInputSingle(
359         IPancakeV3Router.ExactInputSingleParams({
360           tokenIn: _token0,
361           tokenOut: _token1,
362           fee: poolFee,
363           recipient: address(this),
364           amountIn: _amountIn0,
365           amountOutMinimum: 0,
366           sqrtPriceLimitX96: LibTickMath.getSqrtRatioAtTick(_tickUpper -
_tickSpacing - 1)
367         })
368       );
369       // Update optimal amount
370       _optimalAmount0 = _amountIn0 + ERC20(_token0).balanceOf(address(this)) -
_token0Before;
371       _optimalAmount1 = _amountIn1 + _amountOut;
372     }
373   } else {
374     if (_amountIn1 > 0) {
375       uint256 _token1Before = ERC20(_token1).balanceOf(address(this));
376       // one for zero swap
377       ERC20(_token1).safeApprove(address(router), _amountIn1);
378       uint256 _amountOut = router.exactInputSingle(
379         IPancakeV3Router.ExactInputSingleParams({
380           tokenIn: _token1,
381           tokenOut: _token0,
382           fee: poolFee,
383           recipient: address(this),
384           amountIn: _amountIn1,
385           amountOutMinimum: 0,
386           sqrtPriceLimitX96: LibTickMath.getSqrtRatioAtTick(_tickLower +
_tickSpacing + 1)
387         })
388       );
389       // Update optimal amount
390       _optimalAmount0 = _amountIn0 + _amountOut;
391       _optimalAmount1 = _amountIn1 + ERC20(_token1).balanceOf(address(this)) -
_token1Before;
392     }
393   }
```

```
394
395   // Also prepare in range if tick is back in range after swap
396   (, _currTick,,,,) = pool.slot0();
397   if (_tickLower <= _currTick && _currTick <= _tickUpper) {
398     return _prepareOptimalTokensForIncreaseInRange(
399       _token0, _token1, _tickLower, _tickUpper, _optimalAmount0,
      _optimalAmount1
400     );
401   }
402 }
```

However, during the swap process, a price limit is set to the result of the `LibTickMath.getSqrtRatioAtTick(_tickUpper - _tickSpacing - 1)` calculation. This can lead to unexpected results in cases where the difference between the position's `_tickLower` and `_tickUpper` is as narrow as the `_tickSpacing` value.

For example, in the `USDT/BUSD` pool where the `_tickSpacing` is set to 1. When liquidity is added with a narrow tick range, `_tickLower = 1` and `_tickUpper = 2`.

So, during the `_prepareOptimalTokensForIncreaseOutOfRange()` function execution in the `PancakeV3Worker` contract, the calculation result of `_tickUpper - _tickSpacing - 1` may be lower than the `_tickLower` state of the platform's position. As a result, the calculated price limit will also be lower than the lower price range of the platform's position.

This situation can lead to the swapping process potentially driving the price down below the defined range of the position and reverting when adding liquidity due to the current price and calculated amount to increase the position being inconsistent, at lines 235 and 247.

PancakeV3Worker.sol

```
222 function increasePosition(uint256 _amountIn0, uint256 _amountIn1) external
    nonReentrant onlyExecutorInScope {
223   // Can't increase position if position not exist. Use `openPosition` instead.
224   if (nftTokenId == 0) {
225     revert PancakeV3Worker_PositionNotExist();
226   }
227
228   // SLOAD
229   ERC20 _token0 = token0;
230   ERC20 _token1 = token1;
231   int24 _tickLower = posTickLower;
232   int24 _tickUpper = posTickUpper;
233
234   // Prepare optimal tokens for adding liquidity
235   (uint256 _amount0Desired, uint256 _amount1Desired) =
    _prepareOptimalTokensForIncrease(
```

```
236       address(_token0), address(_token1), _tickLower, _tickUpper, _amountIn0,
      _amountIn1
237     );
238
239     // Increase existing position liquidity
240     // SLOAD
241     IPancakeV3MasterChef _masterChef = masterChef;
242     uint256 _nftTokenId = nftTokenId;
243
244     _token0.safeApprove(address(_masterChef), _amount0Desired);
245     _token1.safeApprove(address(_masterChef), _amount1Desired);
246     (, uint256 _amount0, uint256 _amount1) = _masterChef.increaseLiquidity(
247       IPancakeV3MasterChef.IncreaseLiquidityParams({
248         tokenId: _nftTokenId,
249         amount0Desired: _amount0Desired,
250         amount1Desired: _amount1Desired,
251         amount0Min: 0,
252         amount1Min: 0,
253         deadline: block.timestamp
254       })
255     );
256
257     emit LogIncreasePosition(_nftTokenId, msg.sender, _tickLower, _tickUpper,
      _amount0, _amount1);
258 }
```

5.6.2. Remediation

Inspex suggests adding a mechanism to ensure that the tick lower and tick upper have a sufficient range to calculate according to the formula described above. For instance, implementing a mechanism to enforce a minimum range for the liquidity position, where the range should be greater than or equal to 2 times the tick spacing. For example, as shown at line 259.

PCSV3Executor01.sol

```
254 function openPosition(int24 _tickLower, int24 _tickUpper, uint256 _amountIn0,
    uint256 _amountIn1)
255   external
256   onlyVaultManager
257 {
258   address _worker = _getCurrentWorker();
259   require(_tickUpper - _tickLower >= 2 *
    PancakeV3Worker(_worker).pool().tickSpacing());
260   PancakeV3Worker(_worker).openPosition(_tickLower, _tickUpper, _amountIn0,
    _amountIn1);
261   emit LogOpenPosition(_getCurrentVaultToken(), _worker, _tickLower,
    _tickUpper, _amountIn0, _amountIn1);
262 }
```

This ensures that the price limit will remain within the specified range when calculated using the formula `_tickUpper - _tickSpacing - 1)` or `_tickLower + _tickSpacing + 1`.

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

inspeX

CYBERSECURITY PROFESSIONAL SERVICE