Alpaca Stablecoin

Smart Contract Audit Report
Prepared for Alpaca Finance

Date Issued: Nov 15, 2021
Project ID: AUDIT2021035
Version: v1.0
Confidentiality Level: Public

inspex
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021035
Version	v1.0
Client	Alpaca Finance
Project	Alpaca Stablecoin
Auditor(s)	Weerawat Pawanawiwat Patipon Suwanbol Suvicha Buakhom Peeraphut Punsuwan
Author	Patipon Suwanbol
Reviewer	Weerawat Pawanawiwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Nov 15, 2021	Full report	Patipon Suwanbol

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 an audit to verify the security posture of the Alpaca Stablecoin smart contracts between Nov 1, 2021 and Nov 8, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Alpaca Stablecoin 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, 2 medium, 1 low, and 3 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved or mitigated in the reassessment. Therefore, Inspex trusts that Alpaca Stablecoin smart contracts have high-level protections in place to be safe from most attacks.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), 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 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 principles and resulting profits.

Alpaca Stablecoin ($AUSD) is a new multi-collateral stablecoin system adapted from Maker Foundation's Multi-Collateral Dai. The system is composed of multiple smart contracts that manage the collateral positions of the users. The users can mint $AUSD by pledging assets as collaterals. The collateral pledged can be liquidated if the value gets below the threshold to ensure that all $AUSD minted is properly collateralized.

This helps the platform users who see the potential of their assets in the future to utilize the value of their assets without selling. In addition, those collaterals can also generate yields from the farming reward through Alpaca's `FairLaunch` smart contract. Hence, the platform users can maximize their earnings - both in the present and in the future.

Scope Information:

Project Name	Alpaca Stablecoin
Website	https://app.alpacafinance.org/
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Nov 1, 2021 - Nov 8, 2021
Reassessment Date	Nov 12, 2021

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

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

2.2. Scope

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

Initial Audit: (Commit: 0b684fcf4deedeba4c02a2454ccfbb10bc1e8f03)

Contract	Location (URL)
FlashMintModule	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/flash-mint/FlashMintModule.sol
GetPositions	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/managers/GetPositions.sol
PositionHandler	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/managers/PositionHandler.sol
PositionManager	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/managers/PositionManager.sol
AlpacaOraclePriceFeed	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/price-feeders/AlpacaOraclePriceFeed.sol
IbTokenPriceFeed	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/price-feeders/IbTokenPriceFeed.sol
SimplePriceFeed	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/price-feeders/SimplePriceFeed.sol
StrictAlpacaOraclePriceFeed	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/price-feeders/StrictAlpacaOraclePriceFeed.sol
DexPriceOracle	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/price-oracles/DexPriceOracle.sol
AlpacaStablecoinProxyActions	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/proxy-actions/AlpacaStablecoinProxyActions.sol
AlpacaAuth	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/proxy-wallet/AlpacaAuth.sol
AlpacaNote	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/proxy-wallet/AlpacaNote.sol
ProxyWallet	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/proxy-wallet/ProxyWallet.sol
ProxyWalletCache	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/proxy-wallet/ProxyWalletCache.sol

ProxyWalletFactory	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/proxy-wallet/ProxyWalletFactory.sol
ProxyWalletRegistry	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/proxy-wallet/ProxyWalletRegistry.sol
IbTokenAdapter	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/adapters/FarmableTokenAdapter/IbTokenAdapter.sol
AuthTokenAdapter	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/adapters/AuthTokenAdapter.sol
StablecoinAdapter	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/adapters/StablecoinAdapter.sol
TokenAdapter	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/adapters/TokenAdapter.sol
AccessControlConfig	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/config/AccessControlConfig.sol
CollateralPoolConfig	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/config/CollateralPoolConfig.sol
FixedSpreadLiquidationStrategy	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/liquidation-strategies/FixedSpreadLiquidationStrategy.sol
AlpacaStablecoin	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/AlpacaStablecoin.sol
BookKeeper	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/BookKeeper.sol
LiquidationEngine	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/LiquidationEngine.sol
PriceOracle	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/PriceOracle.sol
ShowStopper	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/ShowStopper.sol
StabilityFeeCollector	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/StabilityFeeCollector.sol
StableSwapModule	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/StableSwapModule.sol
SystemDebtEngine	https://github.com/alpaca-finance/alpaca-stablecoin/blob/0b684fcf4d/contracts/6.12/stablecoin-core/SystemDebtEngine.sol

Reassessment: (Commit: 71a7c7b722fa65541cb11a38efec6bc3aef15f18)

Contract	Location (URL)
FlashMintModule	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/flash-mint/FlashMintModule.sol
GetPositions	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/managers/GetPositions.sol
PositionHandler	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/managers/PositionHandler.sol
PositionManager	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/managers/PositionManager.sol
AlpacaOraclePriceFeed	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/price-feeders/AlpacaOraclePriceFeed.sol
IbTokenPriceFeed	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/price-feeders/IbTokenPriceFeed.sol
SimplePriceFeed	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/price-feeders/SimplePriceFeed.sol
StrictAlpacaOraclePriceFeed	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/price-feeders/StrictAlpacaOraclePriceFeed.sol
DexPriceOracle	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/price-oracles/DexPriceOracle.sol
AlpacaStablecoinProxyActions	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/proxy-actions/AlpacaStablecoinProxyActions.sol
AlpacaAuth	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/proxy-wallet/AlpacaAuth.sol
AlpacaNote	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/proxy-wallet/AlpacaNote.sol
ProxyWallet	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/proxy-wallet/ProxyWallet.sol
ProxyWalletCache	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/proxy-wallet/ProxyWalletCache.sol
ProxyWalletFactory	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/proxy-wallet/ProxyWalletFactory.sol
ProxyWalletRegistry	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contract

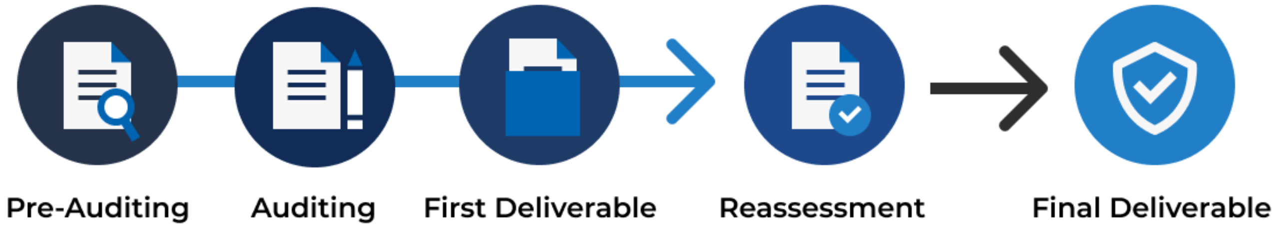
	s/6.12/proxy-wallet/ProxyWalletRegistry.sol
IbTokenAdapter	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/adapters/FarmableTokenAdapter/IbTokenAdapter.sol
AuthTokenAdapter	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/adapters/AuthTokenAdapter.sol
StablecoinAdapter	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/adapters/StablecoinAdapter.sol
TokenAdapter	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/adapters/TokenAdapter.sol
AccessControlConfig	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/config/AccessControlConfig.sol
CollateralPoolConfig	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/config/CollateralPoolConfig.sol
FixedSpreadLiquidationStrategy	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/liquidation-strategies/FixedSpreadLiquidationStrategy.sol
AlpacaStablecoin	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/AlpacaStablecoin.sol
BookKeeper	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/BookKeeper.sol
LiquidationEngine	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/LiquidationEngine.sol
PriceOracle	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/PriceOracle.sol
ShowStopper	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/ShowStopper.sol
StabilityFeeCollector	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/StabilityFeeCollector.sol
StableSwapModule	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/StableSwapModule.sol
SystemDebtEngine	https://github.com/alpaca-finance/alpaca-stablecoin/blob/71a7c7b722/contracts/6.12/stablecoin-core/SystemDebtEngine.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 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
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
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

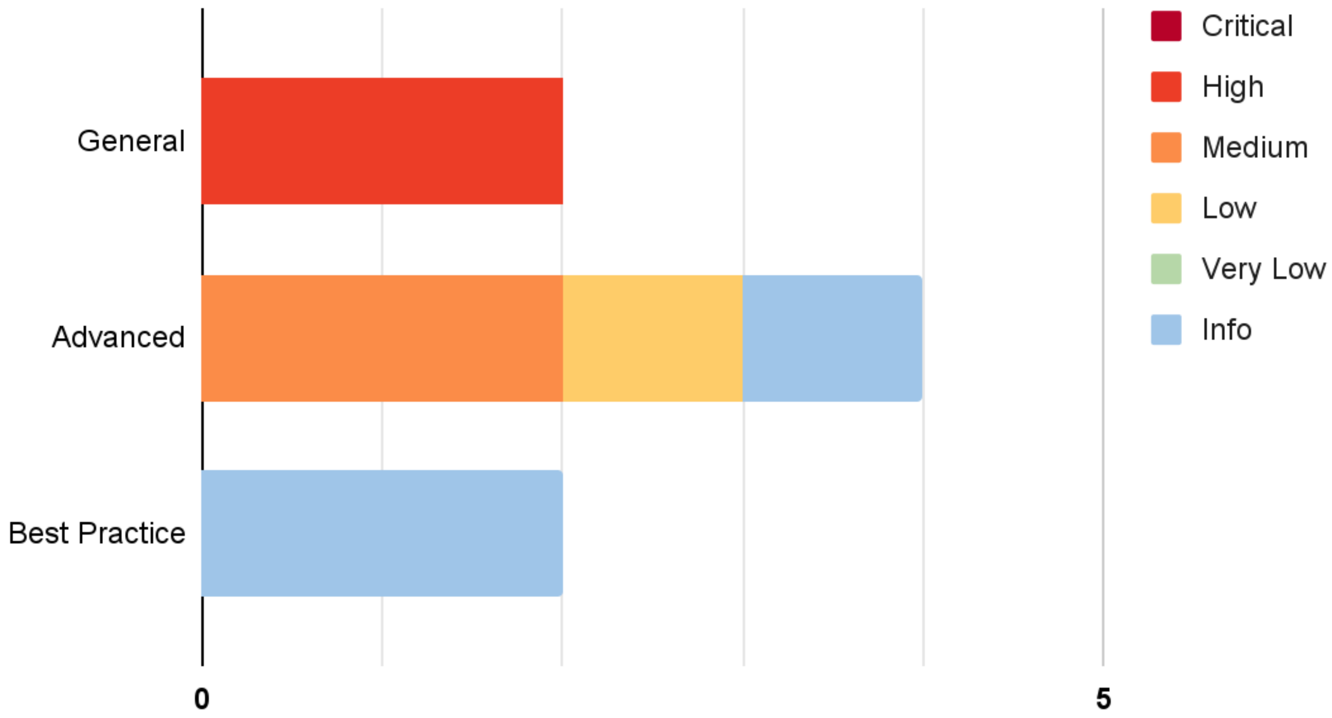
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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 8 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

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

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

ID	Title	Category	Severity	Status
IDX-001	Use of Upgradable Contract Design	General	High	Resolved *
IDX-002	Centralized Control of State Variables	General	High	Resolved *
IDX-003	Improper Pausing for Emergency Function	Advanced	Medium	Resolved
IDX-004	Unsynced Collateral and Staking Balance	Advanced	Medium	Resolved
IDX-005	Unsafe Price Oracle	Advanced	Low	Resolved
IDX-006	Unchecked treasuryFeeBps Value Initialization	Advanced	Info	Resolved
IDX-007	Improper Function Visibility	Best Practice	Info	Resolved
IDX-008	Unused Dependency	Best Practice	Info	Resolved

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

5. Detailed Findings Information

5.1 Use of Upgradable Contract Design

ID	IDX-001
Target	AccessControlConfig AlpacaOraclePriceFeed AlpacaStablecoin AuthTokenAdapter BookKeeper CollateralPoolConfig DexPriceOracle FixedSpreadLiquidationStrategy FlashMintModule IbTokenAdapter IbTokenPriceFeed LiquidationEngine PositionManager PriceOracle ProxyWalletRegistry ShowStopper SimplePriceFeed StabilityFeeCollector StableSwapModule StablecoinAdapter StrictAlpacaOraclePriceFeed SystemDebtEngine TokenAdapter
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The logic of affected contracts can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the users' funds anytime they want. **Likelihood:** Medium This action can be performed by the proxy owner without any restriction.
Status	Resolved * Alpaca Finance team has confirmed that the upgradable contracts will be upgraded through the `Timelock` contract. This means any action that would occur to the upgradeable contracts will be able to be monitored by the community conveniently.

	However, as the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are in effect of the `Timelock` contract before using them.

5.1.1 Description

Smart contracts are designed to be used as agreements that cannot be changed forever. When a smart contract is upgraded, the agreement can be changed from what was previously agreed upon.

As these smart contracts are upgradable, the logic of them can be modified by the owner anytime, making the smart contracts untrustworthy.

5.1.2 Remediation

Inspex suggests deploying the contracts without the proxy pattern or any solution that can make smart contracts upgradeable.

However, if the upgradability is needed, Inspex suggests mitigating this issue by implementing a timelock mechanism with a sufficient length of time to delay the changes. This allows the platform users to monitor the timelock and is notified of the potential changes being done on the smart contracts.

5.2 Centralized Control of State Variables

ID	IDX-002
Target	AlpacaOraclePriceFeed BookKeeper CollateralPoolConfig FlashMintModule PositionManager PriceOracle ShowStopper SimplePriceFeed StabilityFeeCollector StableSwapModule SystemDebtEngine StrictAlpacaOraclePriceFeed
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, the changes are limited by fixed values in the smart contracts.
Status	Resolved * Alpaca Finance team has confirmed that the contracts will be under the `Timelock` contract as same as other contracts on Alpaca Finance. This means all critical state variables will be able to be monitored with delay though the `Timelock` contract. However, as the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are under the `Timelock` contract before using them. Furthermore, all these critical state variables might be handled through the Alpaca Governance to let the community decide the platform's critical actions in the future.

5.2.1 Description

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

However, 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:

File	Contract	Function	Modifier / Role
AlpacaOraclePriceFeed.sol (L:53)	AlpacaOraclePriceFeed	setPriceLife()	onlyOwner
BookKeeper.sol (L:152)	BookKeeper	setTotalDebtCeiling()	OWNER_ROLE
BookKeeper.sol (L:160)	BookKeeper	setAccessControlConfig()	OWNER_ROLE
BookKeeper.sol (L:178)	BookKeeper	setCollateralPoolConfig()	OWNER_ROLE
CollateralPoolConfig.sol (L:124)	CollateralPoolConfig	setDebtCeiling()	onlyOwner
CollateralPoolConfig.sol (L:129)	CollateralPoolConfig	setDebtFloor()	onlyOwner
CollateralPoolConfig.sol (L:134)	CollateralPoolConfig	setPriceFeed()	onlyOwner
CollateralPoolConfig.sol (L:139)	CollateralPoolConfig	setLiquidationRatio()	onlyOwner
CollateralPoolConfig.sol (L:170)	CollateralPoolConfig	setStabilityFeeRate()	onlyOwner
CollateralPoolConfig.sol (L:176)	CollateralPoolConfig	setAdapter()	onlyOwner
CollateralPoolConfig.sol (L:181)	CollateralPoolConfig	setCloseFactorBps()	onlyOwner
CollateralPoolConfig.sol (L:187)	CollateralPoolConfig	setLiquidatorIncentiveBps()	onlyOwner
CollateralPoolConfig.sol (L:196)	CollateralPoolConfig	setTreasuryFeesBps()	onlyOwner
CollateralPoolConfig.sol (L:214)	CollateralPoolConfig	setStrategy()	onlyOwner
FlashMintModule.sol (L:88)	FlashMintModule	setMax()	onlyOwner
FlashMintModule.sol (L:94)	FlashMintModule	setFeeRate()	onlyOwner
PositionManager.sol (L:135)	PositionManager	allowManagePosition()	onlyOwner
PositionManager.sol (L:186)	PositionManager	give()	onlyOwner
PositionManager.sol (L:237)	PositionManager	adjustPosition()	onlyOwner
PositionManager.sol (L:268)	PositionManager	moveCollateral()	onlyOwner
PositionManager.sol (L:287)	PositionManager	moveCollateral()	onlyOwner
PositionManager.sol (L:303)	PositionManager	moveStablecoin()	onlyOwner

PositionManager.sol (L:315)	PositionManager	exportPosition()	onlyOwner
PositionManager.sol (L:341)	PositionManager	importPosition()	onlyOwner
PositionManager.sol (L:366)	PositionManager	movePosition()	onlyOwner
PositionManager.sol (L:413)	PositionManager	redeemLockedCollateral()	onlyOwner
PriceOracle.sol (L:77)	PriceOracle	setStableCoinReferencePrice()	OWNER_ROLE
ShowStopper.sol (L:228)	ShowStopper	setBookKeeper()	OWNER_ROLE
ShowStopper.sol (L:238)	ShowStopper	setLiquidationEngine()	OWNER_ROLE
ShowStopper.sol (L:246)	ShowStopper	setSystemDebtEngine()	OWNER_ROLE
ShowStopper.sol (L:254)	ShowStopper	setPriceOracle()	OWNER_ROLE
ShowStopper.sol (L:262)	ShowStopper	setCageCoolDown()	OWNER_ROLE
SimplePriceFeed.sol (L:49)	SimplePriceFeed	setPrice()	onlyOwner
SimplePriceFeed.sol (L:55)	SimplePriceFeed	setPriceLife()	onlyOwner
StabilityFeeCollector.sol (L:130)	StabilityFeeCollector	setGlobalStabilityFeeRate()	OWNER_ROLE
StabilityFeeCollector.sol (L:137)	StabilityFeeCollector	setSystemDebtEngine()	OWNER_ROLE
StableSwapModule.sol (L:86)	StableSwapModule	setFeeIn()	OWNER_ROLE
StableSwapModule.sol (L:94)	StableSwapModule	setFeeOut()	OWNER_ROLE
SystemDebtEngine.sol (L:62)	SystemDebtEngine	withdrawCollateralSurplus()	OWNER_ROLE
SystemDebtEngine.sol (L:75)	SystemDebtEngine	withdrawStablecoinSurplus()	OWNER_ROLE
SystemDebtEngine.sol (L:89)	SystemDebtEngine	setSurplusBuffer()	OWNER_ROLE
SystemDebtEngine.sol (L:75)	SystemDebtEngine	withdrawStablecoinSurplus()	OWNER_ROLE
SystemDebtEngine.sol (L:62)	SystemDebtEngine	withdrawCollateralSurplus()	OWNER_ROLE
SystemDebtEngine.sol (L:75)	SystemDebtEngine	withdrawStablecoinSurplus()	OWNER_ROLE
StrictAlpacaOraclePriceFeed.sol (L:90)	StrictAlpacaOraclePriceFeed	setPriceLife()	onlyOwner
StrictAlpacaOraclePriceFeed.sol (L:95)	StrictAlpacaOraclePriceFeed	setMaxPriceDiff()	onlyOwner

5.2.2 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 governance to control the use of these functions or mitigate this issue by using a `Timelock` contract to delay the changes for a reasonable amount of time.

5.3 Improper Pausing for Emergency Function

ID	IDX-003
Target	ibTokenAdapter
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** High Users are unable to withdraw their funds in the emergency case when the smart contract is paused by the owner, which results in loss of funds for the users. **Likelihood:** Low Only the owner or the governance can pause the smart contract, and there is no direct benefit for them to gain for pausing, resulting in low motivation for the action.
Status	Resolved Alpaca Finance team has resolved this issue as suggested in commit `713f40573c7a452a82b4c2bbdaf5ce2f443b7b54` (PR #110).

5.3.1 Description

The `ibTokenAdapter` contract is used for handling the user's collateral, in this case, `ibToken`, to the Alpaca's `FairLaunch` contract. This allows the users who hold the $AUSD position to earn $ALPACA reward.

The platform offers a way for the users to withdraw their funds immediately in case of emergency through the `emergencyWithdraw()` function.

ibTokenAdapter.sol

```
/// @dev EMERGENCY ONLY. Withdraw ibToken from FairLaunch with invoking
"_harvest"
function emergencyWithdraw(address _to) external nonReentrant whenNotPaused {
    if (live == 1) {
        uint256 _amount = bookKeeper.collateralToken(collateralPoolId,
msg.sender);
        fairlaunch.withdraw(address(this), pid, _amount);
    }
    _emergencyWithdraw(_to);
}
```

However, the `whenNotPaused` modifier is applied, allowing the authorized account to suspend the execution of the `emergencyWithdraw()` function.

@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol

```
modifier whenNotPaused() {
    require(!paused(), "Pausable: paused");
    _;
}
```

Hence, the platform users will not be able to withdraw their funds in the emergency case when the authorized account pauses the usage of the contract, even when their tokens are not used as the collateral to mint $AUSD.

5.3.2 Remediation

Inspex suggests removing the `whenNotPaused` modifier to allow any user to execute the `emergencyWithdraw()` function during the pause of contract usage.

5.4 Unsynced Collateral and Staking Balance

ID	IDX-004
Target	PositionManager
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Low The staking balance of the moved positions can be mismatched from the balances in the `BookKeeper` contract, causing the farming rewards of the affected positions to be incorrectly distributed. This results in monetary impact on the users and reputation damage to the platform. The balances can be adjusted by manually calling the `moveStake()` function to make the balances matched. **Likelihood:** High The functions related to the transfer of collateral tokens are used in the core functionalities of the system, so they are likely to be called.
Status	Resolved Alpaca Finance team has resolved this issue as suggested in commit `c938e91f1e713c59b4568a9c46c64ad490ec354b` (PR #110).

5.4.1 Description

The `PositionManager` contract provides a list of functions that supports the position adjustments for the users. Most of the provided functions will call the `onMoveCollateral()` of `ibTokenAdapter` internally, which handles the staking collateral amount of the users when the collateral is moved from one place to another.

ibTokenAdapter.sol

```
function onMoveCollateral(
    address _source,
    address _destination,
    uint256 _share,
    bytes calldata _data
) external override nonReentrant whenNotPaused {
    _deposit(_source, 0, _data);
    _moveStake(_source, _destination, _share, _data);
}
```

The `_deposit()` function with `_amount` set as 0 handles the reward harvest for the users.

ibTokenAdapter.sol

```
/// @dev Harvest rewardTokens and distribute to user,
/// deposit collateral tokens to staking contract, and update BookKeeper
/// @param _positionAddress The position address to be updated
/// @param _amount The amount to be deposited
function _deposit(
    address _positionAddress,
    uint256 _amount,
    bytes calldata /* _data */
) private {
    require(live == 1, "IbTokenAdapter/not live");

    harvest(_positionAddress);

    if (_amount > 0) {
        uint256 _share = wdiv(mul(_amount, to18ConversionFactor), netAssetPerShare()); // [wad]
        // Overflow check for int256(wad) cast below
        // Also enforces a non-zero wad
        require(int256(_share) > 0, "IbTokenAdapter/share-overflow");
        address(collateralToken).safeTransferFrom(msg.sender, address(this), _amount);
        bookKeeper.addCollateral(collateralPoolId, _positionAddress, int256(_share));
        totalShare = add(totalShare, _share);
        stake[_positionAddress] = add(stake[_positionAddress], _share);
    }
    rewardDebts[_positionAddress] = rmulup(stake[_positionAddress], accRewardPerShare);

    if (_amount > 0) fairlaunch.deposit(address(this), pid, _amount);

    emit LogDeposit(_amount);
}
```

The `_moveStake()` function handles the transfer of staking amounts of the positions.

ibTokenAdapter.sol

```
/// @dev Move wad amount of staked balance from source to destination.
/// Can only be moved if underlaying assets make sense.
/// @param _source The address to be moved staked balance from
/// @param _destination The address to be moved staked balance to
/// @param _share The amount of staked balance to be moved
function _moveStake(
    address _source,
    address _destination,
    uint256 _share,
```

```
    bytes calldata /* data */
) private {
    // 1. Update collateral tokens for source and destination
    uint256 _stakedAmount = stake[_source];
    stake[_source] = sub(_stakedAmount, _share);
    stake[_destination] = add(stake[_destination], _share);
    // 2. Update source's rewardDebt due to collateral tokens have
    // moved from source to destination. Hence, rewardDebt should be updated.
    // rewardDebtDiff is how many rewards has been paid for that share.
    uint256 _rewardDebt = rewardDebts[_source];
    uint256 _rewardDebtDiff = mul(_rewardDebt, _share) / _stakedAmount;
    // 3. Update rewardDebts for both source and destination
    // Safe since rewardDebtDiff <= rewardDebts[source]
    rewardDebts[_source] = _rewardDebt - _rewardDebtDiff;
    rewardDebts[_destination] = add(rewardDebts[_destination],
_rewardDebtDiff);
    // 4. Sanity check.
    // - stake[source] must more than or equal to collateral + lockedCollateral
that source has
    // to prevent a case where someone try to steal stake from source
    // - stake[destination] must less than or eqal to collateral +
lockedCollateral that destination has
    // to prevent destination from claim stake > actual collateral that he has
    (uint256 _lockedCollateral, ) = bookKeeper.positions(collateralPoolId,
_source);
    require(
        stake[_source] >= add(bookKeeper.collateralToken(collateralPoolId,
_source), _lockedCollateral),
        "IbTokenAdapter/stake[source] < collateralTokens + lockedCollateral"
    );
    (_lockedCollateral, ) = bookKeeper.positions(collateralPoolId,
_destination);
    require(
        stake[_destination] <= add(bookKeeper.collateralToken(collateralPoolId,
_destination), _lockedCollateral),
        "IbTokenAdapter/stake[destination] > collateralTokens +
lockedCollateral"
    );
    emit LogMoveStake(_source, _destination, _share);
}
```

However, some position adjustment functions do not call the `onMoveCollateral()` function to update the staking amount of the positions to be up-to-date, causing the staking balance in the `ibTokenAdapter` to be mismatched, so the reward is incorrectly distributed for the positions moved.

For example, calling the `harvest()` function, the reward amount will be affected by the amount of `stake` as in line 239.

ibTokenAdapter.sol

```
/// @dev Harvest rewards for "_positionAddress" and send to "to"
/// @param _positionAddress The position address that is owned and staked the collateral tokens
function harvest(address _positionAddress) internal {
    // 1. Define the address to receive the harvested rewards
    // Give the rewards to the proxy wallet that owns this position address if there is any
    address _harvestTo = positionManager.mapPositionHandlerToOwner(_positionAddress);
    // if the position owner is not recognized by the position manager,
    // check if the msg.sender is the owner of this position and harvest to msg.sender.
    // or else, harvest to _positionAddress
    if (_harvestTo == address(0)) _harvestTo = msg.sender == _positionAddress ? msg.sender : _positionAddress;
    require(_harvestTo != address(0), "IbTokenAdapter/harvest-to-address-zero");
    // 2. Perform actual harvest. Calculate the new accRewardPerShare.
    if (totalShare > 0) accRewardPerShare = add(accRewardPerShare, rdiv(_harvest(), totalShare));
    // 3. Calculate the rewards that "to" should get by:
    // stake[_positionAddress] * accRewardPerShare (rewards that each share should get) - rewardDebts (what already paid)
    uint256 _rewardDebt = rewardDebts[_positionAddress];
    uint256 _rewards = rmul(stake[_positionAddress], accRewardPerShare);
    if (_rewards > _rewardDebt) {
        uint256 _back = sub(_rewards, _rewardDebt);
        uint256 _treasuryFee = div(mul(_back, treasuryFeeBps), 10000);
        address(rewardToken).safeTransfer(treasuryAccount, _treasuryFee);
        address(rewardToken).safeTransfer(_harvestTo, sub(_back, _treasuryFee));
    }

    // 3. Update accRewardBalance
    accRewardBalance = rewardToken.balanceOf(address(this));
}
```

Hence, the reward will be incorrectly calculated due to the un-updated `stake` values from not calling the `onMoveCollateral()` function.

There are 3 functions that move the collateral between positions but do not invoke the `onMoveCollateral()` function, which are:

- exportPosition()
- importPosition()
- movePosition()

5.4.2 Remediation

Inspex suggests modifying the smart contract to call the `onMoveCollateral()` function in every location that moves the collateral balance from one position to another to keep the staking balance in sync.

5.5 Unsafe Price Oracle

ID	IDX-005
Target	DexPriceOracle
Category	Advanced Smart Contract Vulnerability
CWE	CWE-807: Reliance on Untrusted Inputs in a Security Decision
Risk	**Severity:** Low **Impact:** Medium The price from the oracle can be manipulated, resulting in frozen positions. This includes opening a new position, closing existing positions, and liquidating current positions. This is because the collateral asset value will be used to calculate the position margin (`priceWithSafetyMargin`), and the transaction will be reverted if the collateral asset price is not safe by comparing the primary and secondary price oracle sources with the price difference threshold. In addition, if the primary price oracle source is unstable, the attacker can also manipulate this secondary price oracle source to confirm the correctness of the unreliable value. **Likelihood:** Low It is unlikely that the collateral asset value will be manipulated and affect the platform since `DexPriceOracle` is a secondary source, and the collateral asset value is retrieved from the primary source, Chainlink. Also, there is no direct benefit for the attacker from drastically manipulating the price, as there is a comparison between the prices between the primary and secondary sources.
Status	Resolved Alpaca Finance team has confirmed that they will use the price oracle from Band Protocol instead of the `DexPriceOracle` contract as the secondary source of reliable price. This means the asset value will be more resilient from price manipulation, so the platform's users can ensure the correctness of the asset value.

5.5.1 Description

The `DexPriceOracle` contract is used as one of the price oracles of the platform, allowing other contracts to retrieve the current token price of the given token pair.

The `getPrice()` function calculates the collateral price by using the reserve of `token0` and `token1` in the pair, resulting in the market price of the given token at the time of the execution.

DexPriceOracle.sol

```
/// @dev Return the wad price of token0/token1, multiplied by 1e18
/// NOTE: (if you have 1 token0 how much you can sell it for token1)
function getPrice(address token0, address token1) external view override
```

```
returns (uint256, uint256) {
    if (token0 == token1) return (1e18, uint64(now));

    (uint256 r0, uint256 r1) = PancakeLibraryV2.getReserves(dexFactory, token0,
token1);
    uint256 price = r0.mul(1e18).div(r1);
    return (price, uint64(now));
}
```

This means the transaction order affects the reserve amounts of `token0` and `token1`, so the price can be manipulated easily. This is especially impactful when combined with flashloan attack to drastically change the reserve amounts in the pool.

However, this price is only used in the `StrictAlpacaOraclePriceFeed` contract as the secondary price source to make sure that the price from the primary price source is not drastically different from the price from AMM.

StrictAlpacaOraclePriceFeed.sol

```
function peekPrice() external view override returns (bytes32, bool) {
    (uint256 primaryPrice, uint256 primaryLastUpdate) =
primary.alpacaOracle.getPrice(primary.token0, primary.token1);
    (uint256 secondaryPrice, uint256 secondaryLastUpdate) =
secondary.alpacaOracle.getPrice(
        secondary.token0,
        secondary.token1
    );

    return (bytes32(primaryPrice), _isPriceOk(primaryPrice, secondaryPrice,
primaryLastUpdate, secondaryLastUpdate));
}
```

Therefore, this oracle cannot reliably be used to check that the price from the primary source is valid. If the price from the primary source is drastically different from the actual market price, an attacker can easily manipulate this secondary source to be close to the wrong price from the primary source, and that price will be seen as a valid price by the smart contracts. Also, the availability of the platform can be disrupted if the price is continuously manipulated to be largely different from the primary price source.

5.5.2 Remediation

Inspex suggests using time-weighted average price oracle[2] instead of directly quoting from the reserves.

5.6 Unchecked treasuryFeeBps Value Initialization

ID	IDX-006
Target	ibTokenAdapter
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Alpaca Finance team has resolved this issue as suggested in commit `82a3eb9f1f0849c7f5fd792a860d6346a29f331b` (PR #110).

5.6.1 Description

In the `ibTokenAdapter` contract, the `treasuryFeeBps` state variable indicates the portion of fee that will be distributed to the platform as basis points (bps). The value can be set through the `setTreasuryFeeBps()` function, and the new value (`_treasuryFeeBps`) will be validated in line 189.

IbTokenAdapter.sol

```
187 function setTreasuryFeeBps(uint256 _treasuryFeeBps) external onlyOwner {
188     require(live == 1, "IbTokenAdapter/not-live");
189     require(_treasuryFeeBps <= 5000, "IbTokenAdapter/bad treasury fee bps");
190     treasuryFeeBps = _treasuryFeeBps;
191 }
```

However, during the contract initialization through the `initialize()` function, the `_treasuryFeeBps` value can be set freely, and can exceed the upper limit defined in the set function (`setTreasuryFeeBps()`).

IbTokenAdapter.sol

```
function initialize(
    address _bookKeeper,
    bytes32 _collateralPoolId,
    address _collateralToken,
    address _rewardToken,
    address _fairlaunch,
    uint256 _pid,
    address _shield,
    address _timelock,
    uint256 _treasuryFeeBps,
```

```
    address _treasuryAccount,
    address _positionManager
) external initializer {
    // 1. Initialized all dependencies
    PausableUpgradeable.__Pausable_init();
    ReentrancyGuardUpgradeable.__ReentrancyGuard_init();

    // 2. Sanity checks
    (address _stakeToken, , , , ) =
IAlpacaFairLaunch(_fairlaunch).poolInfo(_pid);
    require(_stakeToken == _collateralToken,
"IbTokenAdapter/collateralToken-not-match");
    require(IAlpacaFairLaunch(_fairlaunch).alpaca() == _rewardToken,
"IbTokenAdapter/reward-token-not-match");
    require(IAlpacaFairLaunch(_fairlaunch).owner() == _shield,
"IbTokenAdapter/shield-not-match");
    require(IShield(_shield).owner() == _timelock,
"IbTokenAdapter/timelock-not-match");

    fairlaunch = IAlpacaFairLaunch(_fairlaunch);
    shield = IShield(_shield);
    timelock = ITimeLock(_timelock);
    pid = _pid;

    live = 1;

    bookKeeper = IBookKeeper(_bookKeeper);
    collateralPoolId = _collateralPoolId;
    collateralToken = _collateralToken;
    decimals = IToken(collateralToken).decimals();
    require(decimals <= 18, "IbTokenAdapter/decimals > 18");

    to18ConversionFactor = 10**(18 - decimals);
    toTokenConversionFactor = 10**decimals;
    rewardToken = IToken(_rewardToken);

    require(_treasuryAccount != address(0), "IbTokenAdapter/bad treasury
account");
    treasuryFeeBps = _treasuryFeeBps;
    treasuryAccount = _treasuryAccount;

    positionManager = IManager(_positionManager);

    address(collateralToken).safeApprove(address(fairlaunch), uint256(-1));
}
```

Hence, without an upper limit checking mechanism in contract initialization, the functions and contracts that rely on the value of the `treasuryFeeBps` state might result in undesirable values or have their transactions reverted. For example, the `_treasuryFee` is calculated using the value of `treasuryFeeBps` in line 242, and it will exceed the amount of reward token available if the bps exceeds 10000, resulting in the transactions being reverted.

ibTokenAdapter.sol

```
223 /// @dev Harvest rewards for "_positionAddress" and send to "to"
224 /// @param _positionAddress The position address that is owned and staked the collateral tokens
225 function harvest(address _positionAddress) internal {
226     // 1. Define the address to receive the harvested rewards
227     // Give the rewards to the proxy wallet that owns this position address if there is any
228     address _harvestTo = positionManager.mapPositionHandlerToOwner(_positionAddress);
229     // if the position owner is not recognized by the position manager,
230     // check if the msg.sender is the owner of this position and harvest to msg.sender.
231     // or else, harvest to _positionAddress
232     if (_harvestTo == address(0)) _harvestTo = msg.sender == _positionAddress ? msg.sender : _positionAddress;
233     require(_harvestTo != address(0), "IbTokenAdapter/harvest-to-address-zero");
234     // 2. Perform actual harvest. Calculate the new accRewardPerShare.
235     if (totalShare > 0) accRewardPerShare = add(accRewardPerShare, rdiv(_harvest(), totalShare));
236     // 3. Calculate the rewards that "to" should get by:
237     // stake[_positionAddress] * accRewardPerShare (rewards that each share should get) - rewardDebts (what already paid)
238     uint256 _rewardDebt = rewardDebts[_positionAddress];
239     uint256 _rewards = rmul(stake[_positionAddress], accRewardPerShare);
240     if (_rewards > _rewardDebt) {
241         uint256 _back = sub(_rewards, _rewardDebt);
242         uint256 _treasuryFee = div(mul(_back, treasuryFeeBps), 10000);
243         address(rewardToken).safeTransfer(treasuryAccount, _treasuryFee);
244         address(rewardToken).safeTransfer(_harvestTo, sub(_back, _treasuryFee));
245     }
246 
247     // 3. Update accRewardBalance
248     accRewardBalance = rewardToken.balanceOf(address(this));
249 }
```

However, the impact is not permanent as the contract owner can use the `setTreasuryFeeBps()` function to set the bps to a proper value.

5.6.2 Remediation

Inspex suggests validating the value of `_treasuryFeeBps` during the contract initialization, for example:

IbTokenAdapter.sol

```
function initialize(
    address _bookKeeper,
    bytes32 _collateralPoolId,
    address _collateralToken,
    address _rewardToken,
    address _fairlaunch,
    uint256 _pid,
    address _shield,
    address _timelock,
    uint256 _treasuryFeeBps,
    address _treasuryAccount,
    address _positionManager
) external initializer {
    // 1. Initialized all dependencies
    PausableUpgradeable.__Pausable_init();
    ReentrancyGuardUpgradeable.__ReentrancyGuard_init();

    // 2. Sanity checks
    (address _stakeToken, , , , ) =
IAlpacaFairLaunch(_fairlaunch).poolInfo(_pid);
    require(_stakeToken == _collateralToken,
"IbTokenAdapter/collateralToken-not-match");
    require(IAlpacaFairLaunch(_fairlaunch).alpaca() == _rewardToken,
"IbTokenAdapter/reward-token-not-match");
    require(IAlpacaFairLaunch(_fairlaunch).owner() == _shield,
"IbTokenAdapter/shield-not-match");
    require(IShield(_shield).owner() == _timelock,
"IbTokenAdapter/timelock-not-match");

    fairlaunch = IAlpacaFairLaunch(_fairlaunch);
    shield = IShield(_shield);
    timelock = ITimeLock(_timelock);
    pid = _pid;

    live = 1;

    bookKeeper = IBookKeeper(_bookKeeper);
    collateralPoolId = _collateralPoolId;
    collateralToken = _collateralToken;
    decimals = IToken(collateralToken).decimals();
    require(decimals <= 18, "IbTokenAdapter/decimals > 18");

    to18ConversionFactor = 10**(18 - decimals);
```

```
    toTokenConversionFactor = 10**decimals;
    rewardToken = IToken(_rewardToken);

    require(_treasuryAccount != address(0), "IbTokenAdapter/bad treasury account");
    require(_treasuryFeeBps <= 5000, "IbTokenAdapter/bad treasury fee bps");
    treasuryFeeBps = _treasuryFeeBps;
    treasuryAccount = _treasuryAccount;

    positionManager = IManager(_positionManager);

    address(collateralToken).safeApprove(address(fairlaunch), uint256(-1));
}
```

5.7 Improper Function Visibility

ID	IDX-007
Target	AlpacaAuth AlpacaStablecoinProxyActions PositionManager ProxyWallet ProxyWalletCache ProxyWalletFactory ProxyWalletRegistry StableSwapModule
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Alpaca Finance team has resolved this issue as suggested in commit `6e24984e6377a34aeeb04dee9969411f8e383829` (PR #110).

5.7.1 Description

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

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

AlpacaStablecoinProxyActions.sol

```
function unlockBNB(
    address _manager,
    address _bnbAdapter,
    uint256 _positionId,
    uint256 _amount, // [wad]
    bytes calldata _data
) public {
    // Unlocks WBNB amount from the CDP
    adjustPosition(_manager, _positionId, -_safeToInt(_amount), 0, _bnbAdapter,
_data);
    // Moves the amount from the CDP positionAddress to proxy's address
    moveCollateral(_manager, _positionId, address(this), _amount, _bnbAdapter,
```

```
_data);
    // Withdraws WBNB amount to proxy address as a token
    IGenericTokenAdapter(_bnbAdapter).withdraw(address(this), _amount, _data);
    // Converts WBNB to BNB
    IWBNB(address(IGenericTokenAdapter(_bnbAdapter).collateralToken()))
.withdraw(_amount);
    // Sends BNB back to the user's wallet
    SafeToken.safeTransferETH(msg.sender, _amount);
}
```

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

File	Contract	Function
AlpacaAuth.sol (L:32)	AlpacaAuth	setOwner()
AlpacaAuth.sol (L:37)	AlpacaAuth	setAuthority()
AlpacaStablecoinProxyActions.sol (L:433)	AlpacaStablecoinProxyActions	unlockBNB()
AlpacaStablecoinProxyActions.sol (L:452)	AlpacaStablecoinProxyActions	unlockToken()
AlpacaStablecoinProxyActions.sol (L:763)	AlpacaStablecoinProxyActions	convertAndLockToken()
AlpacaStablecoinProxyActions.sol (L:776)	AlpacaStablecoinProxyActions	convertLockTokenAndDraw()
AlpacaStablecoinProxyActions.sol (L:802)	AlpacaStablecoinProxyActions	convertBNBAndLockToken()
AlpacaStablecoinProxyActions.sol (L:814)	AlpacaStablecoinProxyActions	convertBNBLockTokenAndDraw()
PositionManager.sol (L:135)	PositionManager	allowManagePosition()
PositionManager.sol (L:147)	PositionManager	allowMigratePosition()
PositionManager.sol (L:155)	PositionManager	open()
PositionManager.sol (L:186)	PositionManager	give()
PositionManager.sol (L:243)	PositionManager	adjustPosition()
PositionManager.sol (L:274)	PositionManager	moveCollateral()

PositionManager.sol (L:294)	PositionManager	moveCollateral()
PositionManager.sol (L:307)	PositionManager	moveStablecoin()
PositionManager.sol (L:316)	PositionManager	exportPosition()
PositionManager.sol (L:342)	PositionManager	importPosition()
PositionManager.sol (L:367)	PositionManager	movePosition()
PositionManager.sol (L:418)	PositionManager	redeemLockedCollateral()
ProxyWallet.sol (L:36)	AlpacaAuth	execute()
ProxyWallet.sol (L:49)	AlpacaAuth	execute()
ProxyWallet.sol (L:71)	AlpacaAuth	setCache()
ProxyWalletCache.sol (L:24)	ProxyWalletCache	write()
ProxyWalletFactory.sol (L:33)	ProxyWalletFactory	build()
ProxyWalletFactory.sol (L:39)	ProxyWalletFactory	build()
ProxyWalletRegistry.sol (L:35)	ProxyWalletRegistry	build()
ProxyWalletRegistry.sol (L:41)	ProxyWalletRegistry	build()
ProxyWalletRegistry.sol (L:47)	ProxyWalletRegistry	setOwner()
StableSwapModule.sol (L:104)	StableSwapModule	whitelist()
StableSwapModule.sol (L:110)	StableSwapModule	blacklist()

5.7.2 Remediation

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

AlpacaStablecoinProxyActions.sol

```
function unlockBNB(
    address _manager,
    address _bnbAdapter,
    uint256 _positionId,
    uint256 _amount, // [wad]
    bytes calldata _data
) external {
    // Unlocks WBNB amount from the CDP
    adjustPosition(_manager, _positionId, -_safeToInt(_amount), 0, _bnbAdapter,
_data);
```

```
    // Moves the amount from the CDP positionAddress to proxy's address
    moveCollateral(_manager, _positionId, address(this), _amount, _bnbAdapter, 
_data);
    // Withdraws WBNB amount to proxy address as a token
    IGenericTokenAdapter(_bnbAdapter).withdraw(address(this), _amount, _data);
    // Converts WBNB to BNB
   IWBNB(address(IGenericTokenAdapter(_bnbAdapter).collateralToken()))
.withdraw(_amount);
    // Sends BNB back to the user's wallet
    SafeToken.safeTransferETH(msg.sender, _amount);
}
```

5.8 Unused Dependency

ID	IDX-008
Target	DexPriceOracle
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved Alpaca Finance team has resolved this issue as suggested in commit `10021eaac4540ebf98c05db9c6bf88c4988a91f5` (PR #110).

5.8.1 Description

The `DexPriceOracle` contract inherits from OpenZeppelin's `PausableUpgradeable` abstract contract, allowing the contract to be suspended from the authorized account.

DexPriceOracle.sol

```
contract DexPriceOracle is PausableUpgradeable, IAlpacaOracle {
```

However, in the `initialize()` function, it does not initialize the inherited `PausableUpgradeable` contract, and the functions and modifiers of the `PausableUpgradeable` are not used anywhere in the `DexPriceOracle` contract.

DexPriceOracle.sol

```
function initialize(address _dexFactory) external initializer {
    dexFactory = _dexFactory;
}
```

Inheriting a contract without any use results in unnecessary gas usage on the deployment of the contract.

5.8.2 Remediation

Inspex suggests removing the unused dependency, in this case, the `PausableUpgradeable` abstract contract to reduce unnecessary gas usage.

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] "TWAP Oracle" [Online]. Available: https://docs.uniswap.org/protocol/V2/concepts/core-concepts/oracles. [Accessed: 01-November-2021]

inspex

CYBERSECURITY PROFESSIONAL SERVICE