Alperp

Smart Contract Audit Report
Prepared for Alpaca Finance

Date Issued:	Mar 9, 2023
Project ID:	AUDIT2023002
Version:	v1.0
Confidentiality Level:	Public

inspex
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2023002
Version	v1.0
Client	Alpaca Finance
Project	Alperp
Auditor(s)	Natsasit Jirathammanuwat Darunphop Pengkumta Wachirawit Kanpanluk
Author(s)	Natsasit Jirathammanuwat Darunphop Pengkumta Wachirawit Kanpanluk
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Mar 9, 2023	Full report	Natsasit Jirathammanuwat Darunphop Pengkumta Wachirawit Kanpanluk

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 Alperp smart contracts between Feb 15, 2023 and Feb 21, 2023. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Alperp 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 4 high, 5 medium, 2 low, 2 very low, and 2 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 Alperp smart contracts have high-level protections in place to be safe from most attacks.

This smart contract passes Inspex's security verification standard, and is trustworthy.
Approved by Inspex on Mar 9, 2023
inspex
CYBERSECURITY PROFESSIONAL SERVICE
PASS

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 Perpetual Trading (Alperp) is an automated perpetual trading protocol that allows liquidity providers to deposit funds into a mixed-asset liquidity pool called the ALP pool. The liquidity providers will receive ALP tokens representing their portion of the pool's liquidity. Traders can utilize the funds in the ALP pool to open perpetual trade positions and gain leverage. In this system, traders and ALP holders are counter-parties, meaning that any losses incurred by traders will be reflected in the value of the ALP token and vice versa.

Scope Information:

Project Name	Alperp
Website	https://www.alpacafinance.org/
Smart Contract Type	Ethereum Smart Contract
Chain	BNB Smart Chain
Programming Language	Solidity
Category	Futures, AMM, Yield Farming

Audit Information:

Audit Method	Whitebox
Audit Date	Feb 15, 2023 - Feb 21, 2023
Reassessment Date	Mar 9, 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: 7c39197db2085b0fd3e7adff558529484531802d)

Contract	Location (URL)
MerkleAirdrop	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/airdrop/MerkleAirdrop.sol
MerkleProof	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/airdrop/MerkleProof.sol
AlpacaVaultFarmStrategy	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/AlpacaVaultFarmStrategy.sol
Constants	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/Constants.sol
PoolOracle	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/PoolOracle.sol
PythPriceFeed	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/PythPriceFeed.sol
WNativeRelayer	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/WNativeRelayer.sol
PoolDiamond	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/PoolDiamond.sol
PoolRouter03	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/PoolRouter03.sol
AccessControlFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/AccessControlFacet.sol
AdminFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/AdminFacet.sol
DiamondCutFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/DiamondCutFacet.sol
DiamondLoupeFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/DiamondLoupeFacet.sol
FarmFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/FarmFacet.sol

FundingRateFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/FundingRateFacet.sol
GetterFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/GetterFacet.sol
LiquidityFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/LiquidityFacet.sol
OwnershipFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/OwnershipFacet.sol
PerpTradeFacet	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/facets/PerpTradeFacet.sol
AccessControlInitializer	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/initializers/AccessControlInitializer.sol
DiamondInitializer	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/initializers/DiamondInitializer.sol
PoolConfigInitializer	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/initializers/PoolConfigInitializer.sol
PoolConfigInitializer02	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/initializers/PoolConfigInitializer02.sol
IterableMapping	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/libraries/IterableMapping.sol
LibAccessControl	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/libraries/LibAccessControl.sol
LibDiamond	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/libraries/LibDiamond.sol
LibPoolConfigV1	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/libraries/LibPoolConfigV1.sol
LibPoolV1	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/libraries/LibPoolV1.sol
LibReentrancyGuard	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/libraries/LibReentrancyGuard.sol
Orderbook02	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/core/pool-diamond/limit-orders/Orderbook02.sol
LinkedList	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/libraries/LinkedList.sol

TransparentUpgradeableProxy	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/proxy/TransparentUpgradeableProxy.sol
ALPStaking	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/staking/ALPStaking.sol
FeedableRewarder	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/staking/FeedableRewarder.sol
RewardDistributor	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/staking/RewardDistributor.sol
TimelockController	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/timelock/TimelockController.sol
ALP	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/tokens/ALP.sol
BaseMintableToken	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/tokens/base/BaseMintableToken.sol
Math	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/utils/Math.sol
Multicall	https://github.com/inspex-archive/Alpaca-Finance_Alperp/blob/7c39197db2/src/utils/Multicall.sol

Reassessment: (Commit: 273f7a69df94d3031a8c811076440a3cfd99bf29)

Contract	Location (URL)
MerkleAirdrop	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/airdrop/MerkleAirdrop.sol
MerkleProof	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/airdrop/MerkleProof.sol
AlpacaVaultFarmStrategy	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/AlpacaVaultFarmStrategy.sol
Constants	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/Constants.sol
PoolOracle	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/PoolOracle.sol
PythPriceFeed	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/PythPriceFeed.sol

WNativeRelayer	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/WNativeRelayer.sol
PoolDiamond	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/PoolDiamond.sol
PoolRouter03	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/PoolRouter03.sol
AccessControlFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/AccessControlFacet.sol
AdminFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/AdminFacet.sol
DiamondCutFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/DiamondCutFacet.sol
DiamondLoupeFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/DiamondLoupeFacet.sol
FarmFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/FarmFacet.sol
FundingRateFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/FundingRateFacet.sol
GetterFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/GetterFacet.sol
LiquidityFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/LiquidityFacet.sol
OwnershipFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/OwnershipFacet.sol
PerpTradeFacet	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/facets/PerpTradeFacet.sol
AccessControlInitializer	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/initializers/AccessControlInitializer.sol
DiamondInitializer	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/initializers/DiamondInitializer.sol
PoolConfigInitializer	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/initializers/PoolConfigInitializer.sol
PoolConfigInitializer02	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/initializers/PoolConfigInitializer02.sol

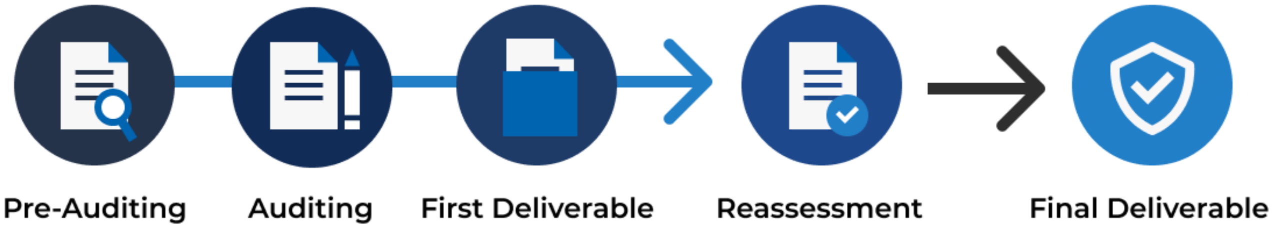
IterableMapping	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/libraries/IterableMapping.sol
LibAccessControl	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/libraries/LibAccessControl.sol
LibDiamond	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/libraries/LibDiamond.sol
LibPoolConfigV1	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/libraries/LibPoolConfigV1.sol
LibPoolV1	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/libraries/LibPoolV1.sol
LibReentrancyGuard	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/libraries/LibReentrancyGuard.sol
Orderbook02	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/core/pool-diamond/limit-orders/Orderbook02.sol
LinkedList	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/libraries/LinkedList.sol
TransparentUpgradeableProxy	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/proxy/TransparentUpgradeableProxy.sol
ALPStaking	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/staking/ALPStaking.sol
FeedableRewarder	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/staking/FeedableRewarder.sol
RewardDistributor	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/staking/RewardDistributor.sol
TimelockController	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/timelock/TimelockController.sol
ALP	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/tokens/ALP.sol
BaseMintableToken	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/tokens/base/BaseMintableToken.sol
Math	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/utils/Math.sol
Multicall	https://github.com/alpaca-finance/alperp-contract/blob/273f7a69df/src/utils/Multicall.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**.

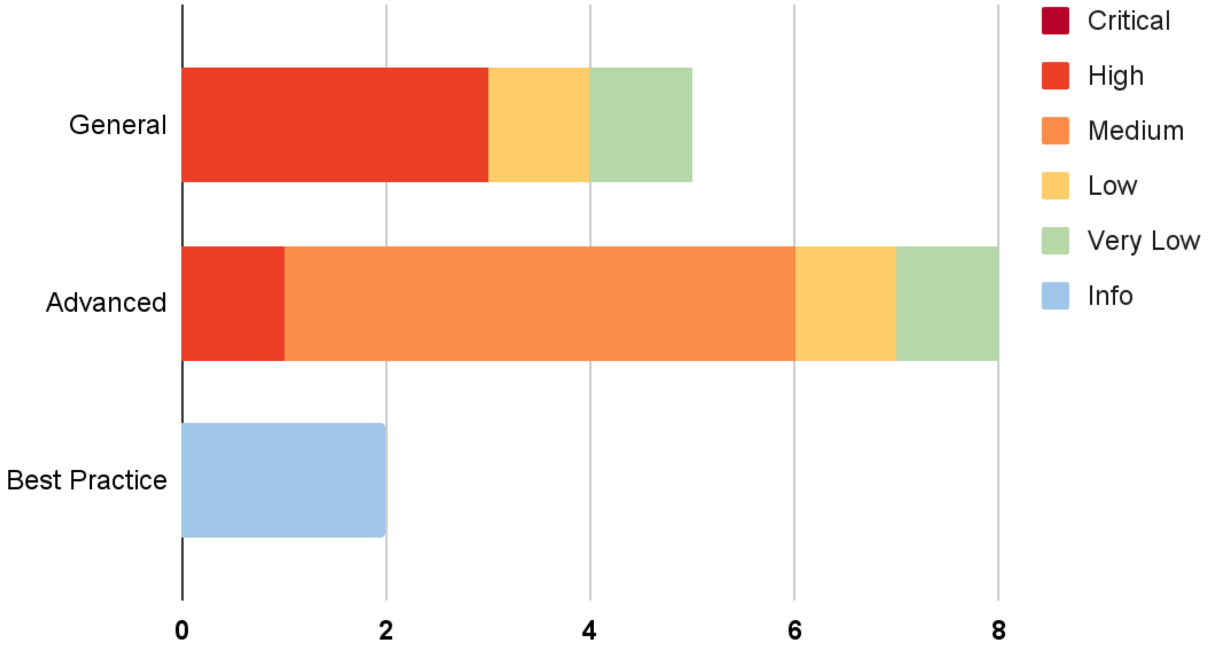
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

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

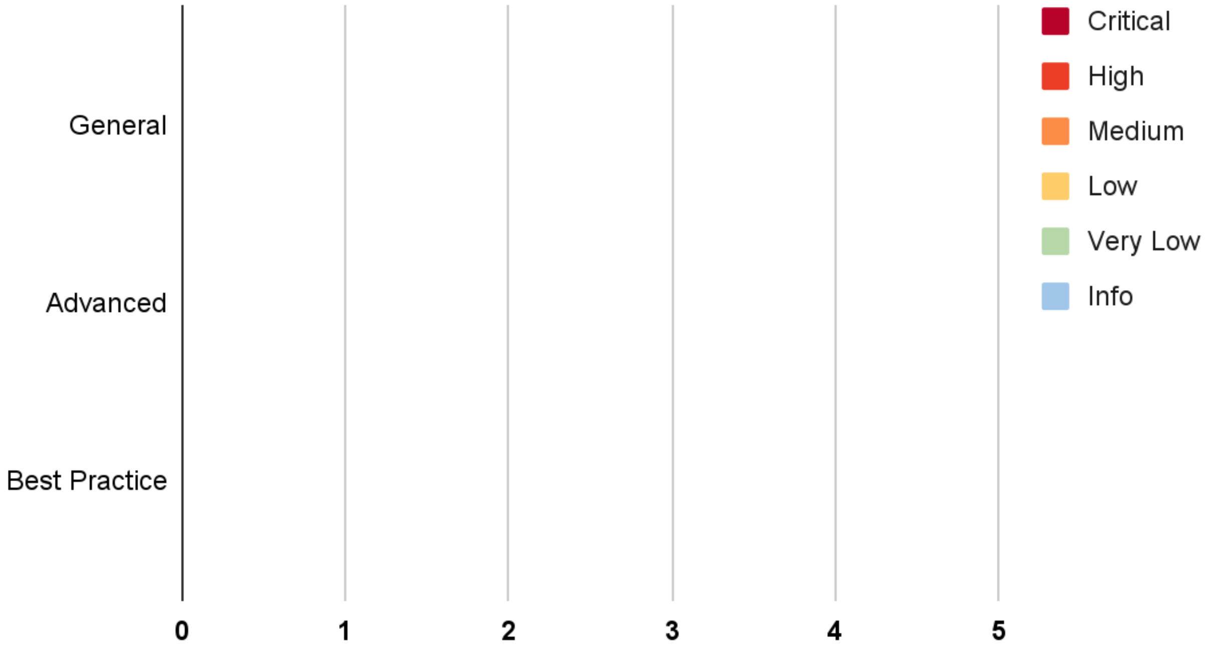
4. Summary of Findings

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	Use of Upgradable Contract Design	General	High	Resolved *
IDX-002	Arbitrary ALP Token Minting	Advanced	High	Resolved *
IDX-003	Centralized Control of State Variable	General	High	Resolved *
IDX-004	External Call to Untrusted Third Party Component	General	High	Resolved *
IDX-005	Miscalculation in getAum() function	Advanced	Medium	Resolved
IDX-006	Denial of Service in PoolRouter03	Advanced	Medium	Resolved
IDX-007	Improper Share Calculation	Advanced	Medium	Resolved
IDX-008	Design Flaw in Fixed Rate Token Swap	Advanced	Medium	Resolved *
IDX-009	Inconsistent swap() function in RewardDistributor	Advanced	Medium	Resolved
IDX-010	Unsafe Token Transfer in RewardDistributor	General	Low	Resolved
IDX-011	Oracle Price Update Bypass	Advanced	Low	Resolved
IDX-012	Missing Input Validation in bulkClaim() Function	Advanced	Very Low	Resolved
IDX-013	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-014	Unnecessary Zero Amount Transfer	Best Practice	Info	Resolved
IDX-015	Unoptimized Invariant Calculation	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	PoolOracle PythPriceFeed PoolDiamond PoolRouter03 AccessControlFacet AdminFacet DiamondCutFacet DiamondLoupeFacet FarmFacet FundingRateFacet GetterFacet LiquidityFacet OwnershipFacet PerpTradeFacet LibPoolV1 Orderbook02 TransparentUpgradeableProxy ALPStaking FeedableRewarder RewardDistributor ALP
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. **Likelihood: Medium** This action can be performed by the proxy owner without any restriction.
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 or upgrades to the contract and also to provide transparency in the process of maintaining contract upgrades. 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

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 the smart contracts upgradeable.

However, if upgradability is needed, Inspex suggests mitigating this issue by implementing a timelock mechanism with a sufficient length of time to delay the changes at least 24 hours on the proxy owner role. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contracts.

5.2. Arbitrary ALP Token Minting

ID	IDX-002
Target	ALP
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** **High** **Impact:** **High** The contract owner can mint unlimited $ALP by adding an arbitrary minter address to the contract. The minted $ALP can be used to remove liquidity from the pool. **Likelihood:** **Medium** Only the contract owner can perform this action.
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 or upgrades to the contract and also to provide transparency in the process of maintaining contract upgrades. 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.

5.2.1. Description

In the `ALP` contract, the `onlyMinter` modifier is an access control modifier that allows to mint the $ALP by executing the `mint()` function with this modifier.

ALP.sol

```
modifier onlyMinter() {
  if (!isMinter[msg.sender]) revert ALP_NotMinter();
  _;
}

```

ALP.sol

```
function mint(address to, uint256 amount) public onlyMinter {
  cooldown[to] = block.timestamp + liquidityCooldown;
  _mint(to, amount);
}
```

Additionally, the contract owner can set any address to be the `onlyMinter` by calling the `setMinter` function.

ALP.sol

```
function setMinter(address minter, bool allow) external onlyOwner {
  isMinter[minter] = allow;
  emit ALP_SetMinter(minter, isMinter[minter], allow);
}
```

As a result, the owner can set any address as the `onlyMinter` modifier, which has the ability to mint an unlimited $ALP.

5.2.2. Remediation

Inspex suggests allowing only the pool to mint the $ALP. This can be done by changing the `onlyMinter` modifier implementation, removing the `setMinter()` function, and setting the minter once in the `initialize()` function as follows:

ALP.sol

```
address public minter;
modifier onlyMinter() {
  if (minter != msg.sender) revert ALP_NotMinter();
  _;
}

function initialize(uint256 liquidityCooldown_, address pool_) external initializer {
  OwnableUpgradeable.__Ownable_init();
  ERC20Upgradeable.__ERC20_init("Alperp Liquidity Provider", "ALP");

  MAX_COOLDOWN_DURATION = 48 hours;
  liquidityCooldown = liquidityCooldown_;
  minter = pool_;
}
```

5.3. Centralized Control of State Variable

ID	IDX-003
Target	MerkleAirdrop PoolOracle PythPriceFeed AccessControlFacet WNativeRelayer AdminFacet FarmFacet OwnershipFacet TransparentUpgradeableProxy Orderbook02 ALPStaking FeedableRewarder RewardDistributor ALP BaseMintableToken
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 or upgrades to contract and also to provide transparency in the process of maintaining contract upgrades. 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.

5.3.1. 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:

File	Contract	Function	Modifier
MerkleAirdrop.sol (L:59)	MerkleAirdrop	setFeeder()	onlyOwner
MerkleAirdrop.sol (L:151)	MerkleAirdrop	emergencyWithdraw()	onlyOwner
PoolOracle.sol (L:68)	PoolOracle	setSecondaryPriceFeed()	onlyOwner
PoolOracle.sol (L:73)	PoolOracle	setIsSecondaryPriceEnabled()	onlyOwner
PoolOracle.sol (L:212)	PoolOracle	setMaxStrictPriceDeviation()	onlyOwner
PoolOracle.sol (L:223)	PoolOracle	setPriceFeed()	onlyOwner
PoolOracle.sol (L:243)	PoolOracle	setRoundDepth()	onlyOwner
PythPriceFeed.sol (L:71)	PythPriceFeed	setTokenPriceId()	onlyOwner
PythPriceFeed.sol (L:83)	PythPriceFeed	setTokenPriceIds()	onlyOwner
PythPriceFeed.sol (L:94)	PythPriceFeed	setMaxPriceAge()	onlyOwner
PythPriceFeed.sol (L:106)	PythPriceFeed	setFavorRefPrice()	onlyOwner
PythPriceFeed.sol (L:113)	PythPriceFeed	setUpdater()	onlyOwner
AccessControlFacet.sol (L:55)	AccessControlFacet	grantRole()	adminRole
AccessControlFacet.sol (L:71)	AccessControlFacet	revokeRole()	adminRole
WNativeRelayer.sol (L:34)	WNativeRelayer	setCallerOk()	onlyOwner
AdminFacet.sol (L:136)	AdminFacet	setPoolOracle()	onlyOwner
AdminFacet.sol (L:148)	AdminFacet	setAllowLiquidators()	onlyOwner

AdminFacet.sol (L:162)	AdminFacet	setFlashLoanFeeBps()	onlyOwner
AdminFacet.sol (L:174)	AdminFacet	setFundingRate()	onlyOwner
AdminFacet.sol (L:209)	AdminFacet	setIsAllowAllLiquidators()	onlyOwner
AdminFacet.sol (L:224)	AdminFacet	setIsDynamicFeeEnable()	onlyOwner
AdminFacet.sol (L:239)	AdminFacet	setIsLeverageEnable()	onlyOwner
AdminFacet.sol (L:251)	AdminFacet	setIsSwapEnable()	onlyOwner
AdminFacet.sol (L:260)	AdminFacet	setLiquidationFeeUsd()	onlyOwner
AdminFacet.sol (L:278)	AdminFacet	setMaxLeverage()	onlyOwner
AdminFacet.sol (L:289)	AdminFacet	setMinProfitDuration()	onlyOwner
AdminFacet.sol (L:304)	AdminFacet	setMintBurnFeeBps()	onlyOwner
AdminFacet.sol (L:316)	AdminFacet	setPositionFeeBps()	onlyOwner
AdminFacet.sol (L:328)	AdminFacet	setRouter()	onlyOwner
AdminFacet.sol (L:337)	AdminFacet	setSwapFeeBps()	onlyOwner
AdminFacet.sol (L:359)	AdminFacet	setTaxBps()	onlyOwner
AdminFacet.sol (L:377)	AdminFacet	setTokenConfigs()	onlyOwner
AdminFacet.sol (L:414)	AdminFacet	setTreasury()	onlyOwner
AdminFacet.sol (L:423)	AdminFacet	deleteTokenConfig()	onlyOwner
AdminFacet.sol (L:458)	AdminFacet	setPlugin()	onlyOwner
FarmFacet.sol (L:73)	FarmFacet	setStrategyOf()	onlyOwner
FarmFacet.sol (L:129)	FarmFacet	setStrategyTargetBps()	onlyOwner
OwnershipFacet.sol (L: 20)	OwnershipFacet	transferOwnership()	contractOwner
TransparentUpgradeableProxy.sol (L: 98)	TransparentUpgradeableProxy	changeAdmin()	ifAdmin
TransparentUpgradeableProxy.sol (L: 107)	TransparentUpgradeableProxy	upgradeTo()	ifAdmin

TransparentUpgradeableProxy.sol (L: 118)	TransparentUpgradeableProxy	upgradeToAndCall()	ifAdmin
Orderbook02.sol (L:301)	Orderbook02	setWhitelist()	onlyOwner
Orderbook02.sol (L:309)	Orderbook02	setIsAllowAllExecutor()	onlyOwner
Orderbook02.sol (L:314)	Orderbook02	setMinExecutionFee()	onlyOwner
Orderbook02.sol (L:320)	Orderbook02	setMinPurchaseTokenAmountUsd()	onlyOwner
ALPStaking.sol (L:56)	ALPStaking	addStakingToken()	onlyOwner
ALPStaking.sol (L:74)	ALPStaking	addRewarder()	onlyOwner
ALPStaking.sol (L:92)	ALPStaking	removeRewarderForTokenByIndex()	onlyOwner
ALPStaking.sol (L:168)	ALPStaking	setCompounder()	onlyOwner
FeedableRewarder.sol (L:167)	FeedableRewarder	setFeeder()	onlyOwner
RewardDistributor.sol (L:138)	RewardDistributor	setParams()	onlyOwner
RewardDistributor.sol (L:184)	RewardDistributor	setReferralRevenueMaxThreshold()	onlyOwner
RewardDistributor.sol (L:198)	RewardDistributor	setFeeder()	onlyOwner
ALP.sol (L:46)	ALP	setLiquidityCooldown()	onlyOwner
ALP.sol (L:57)	ALP	setWhitelist()	onlyOwner
ALP.sol (L:63)	ALP	setMinter()	onlyOwner
BaseMintableToken.sol (L:50)	BaseMintableToken	setMinter()	onlyOwner
BaseMintableToken.sol (L:69)	BaseMintableToken	setMaxSupply()	onlyOwner

5.3.2. 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).

5.4. External Call to Untrusted Third Party Component

ID	IDX-004
Target	FarmFacet
Category	General Smart Contract Vulnerability
CWE	CWE-829: Inclusion of Functionality from Untrusted Control Sphere
Risk	**Severity:** High **Impact:** High An untrusted third-party smart contract may harm the user's funds that are deposited by strategy. **Likelihood:** Medium The vault address of an untrusted smart contract can be set in strategy when deploying a strategy contract, which is controlled by the contract owner. It is possible that a third party who owns an untrusted contract may perform something malicious, intentionally or unintentionally.
Status	Resolved * The Alpaca Finance team has mitigated this issue by committing that only the Alpaca Staking Vault would be set as the strategy contract, along with a pending strategy mechanism that delays the strategy contract from being used, by implementing the Timelock contract as the owner of all contracts to prevent immediate changes or upgrades to contracts, including the setting of the strategy of the platform.

5.4.1. Description

The `FarmFacet` contract has the `setStrategyOf()` function that could set the strategy address contract to a third party vault address.

FarmFacet.sol

```
function setStrategyOf(address token, StrategyInterface newStrategy)
  external
  onlyOwner
  nonReentrant
{
  // Load PoolConfig Diamond storage
  LibPoolConfigV1.PoolConfigV1DiamondStorage
    storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();

  LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
    .strategyDataOf[token];
  StrategyInterface pendingStrategy = poolConfigDs.pendingStrategyOf[token];
```

```
 85     if (strategyData.startTimestamp == 0 || pendingStrategy != newStrategy) {
 86       // When adding new strategy or changing strategy
 87       poolConfigDs.pendingStrategyOf[token] = newStrategy;
 88       emit SetPendingStrategy(token, newStrategy);
 89       strategyData.startTimestamp = uint64(block.timestamp + STRATEGY_DELAY);
 90     } else {
 91       // When committing a new strategy
 92       if (
 93         strategyData.startTimestamp == 0 ||
 94         block.timestamp < strategyData.startTimestamp
 95       ) revert FarmFacet_TooEarlyToCommitStrategy();
 96       if (address(poolConfigDs.strategyOf[token]) != address(0)) {
 97         // If there is previous strategy, we need to withdraw all funds from it
 98         int256 balanceChange = poolConfigDs.strategyOf[token].exit(
 99           strategyData.principle
100         );
101         // Update totalOf[token] to sync physical balance with pool state
102         LibPoolV1.updateTotalOf(token);
103         // Realized profits/losses
104         if (balanceChange > 0) {
105           uint256 profit = uint256(balanceChange);
106           LibPoolV1.increasePoolLiquidity(token, profit);
107
108           emit StrategyRealizedProfit(token, profit);
109         } else if (balanceChange < 0) {
110           uint256 loss = uint256(-balanceChange);
111           LibPoolV1.decreasePoolLiquidity(token, loss);
112
113           emit StrategyRealizedLoss(token, loss);
114         }
115
116         emit StrategyDivest(token, strategyData.principle);
117       }
118       // Commit new strategy
119       poolConfigDs.strategyOf[token] = newStrategy;
120       strategyData.startTimestamp = 0;
121       strategyData.principle = 0;
122       poolConfigDs.pendingStrategyOf[token] = StrategyInterface(address(0));
123
124       emit SetStrategy(token, newStrategy);
125     }
126     poolConfigDs.strategyDataOf[token] = strategyData;
127   }
```

When the `farm()` function is called, the user's funds will be transferred to the strategy contract in line 193, and then the `run()` function call will deploy funds to the vault contract that is controlled by a third party.

FarmFacet.sol

```
function farm(address token, bool isRebalanceNeeded)
  external
  onlyPoolDiamondOrFarmKeeper
{
  // Load PoolV1 diamond storage
  LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
    .poolV1DiamondStorage();

  // Load PoolConfig Diamond storage
  LibPoolConfigV1.PoolConfigV1DiamondStorage
    storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();

  // Load relevant variables
  LibPoolConfigV1.StrategyData memory strategyData = poolConfigDs
    .strategyDataOf[token];
  StrategyInterface strategy = poolConfigDs.strategyOf[token];

  // Realized profits or losses from strategy
  int256 balanceChange = strategy.realized(strategyData.principle);
  // If there is no change in balance, and does not need to rebalance, then stop it here.
  if (balanceChange == 0 && !isRebalanceNeeded) return;

  if (balanceChange > 0) {
    // If there is a profit, then increase pool liquidity
    uint256 profits = uint256(balanceChange);
    LibPoolV1.increasePoolLiquidity(token, profits);

    LibPoolV1.updateTotalOf(token);

    emit StrategyRealizedProfit(token, profits);
  } else if (balanceChange < 0) {
    // If there is a loss, then decrease pool liquidity
    uint256 losses = uint256(-balanceChange);
    LibPoolV1.decreasePoolLiquidity(token, losses);
    strategyData.principle -= losses.toUint128();

    emit StrategyRealizedLoss(token, losses);
  }

  // If rebalance to make sure the strategy has the right amount of funds to deploy, then do it.
  if (isRebalanceNeeded) {
    // Calculate the target amount of funds to be deployed
    uint256 targetDeployedFunds = ((poolV1ds.liquidityOf[token] -
      poolV1ds.reservedOf[token]) * strategyData.targetBps) / 10000;

    if (strategyData.principle < targetDeployedFunds) {
      // If strategy short of funds, then deposit more funds
      // Find out how much more funds to deposit
      uint256 amountOut = targetDeployedFunds - strategyData.principle;

      // Transfer funds from pool to strategy and run it
      LibPoolV1.pushTokens(token, address(strategy), amountOut);
      strategy.run(amountOut);

      // Update how much pool put in the strategy
      strategyData.principle += amountOut.toUint128();

      emit StrategyInvest(token, amountOut);
    } else if (strategyData.principle > targetDeployedFunds) {
      // If strategy has more funds than it should be, then withdraw some funds
      // Find out how much funds to withdraw
      uint256 amountIn = strategyData.principle - targetDeployedFunds;

      // Withdraw funds from strategy and transfer it back to pool
      uint256 actualAmountIn = strategy.withdraw(amountIn);

      // Update how much pool put in the strategy
      strategyData.principle -= actualAmountIn.toUint128();

      LibPoolV1.updateTotalOf(token);

      emit StrategyDivest(token, actualAmountIn);
    }
  }

  poolConfigDs.strategyDataOf[token] = strategyData;
}
```

The risk of the platform funds will depend on the external third-party smart contract or platform risk that the strategy contract interacts with.

5.4.2. Remediation

Inspex suggests performing external calls to only known and trusted smart contracts and avoiding the use of unreliable external third-party smart contracts for managing the platform's funds.

5.5. Miscalculation in getAum() function

ID	IDX-005
Target	GetterFacet
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The `getAum()` function may return a slightly lower value when there are short positions opened in the platform. **Likelihood:** Medium The asset under management's value will be miscalculated when it is calculated with the minimum price from the price oracle. The `getAum()` function with minimum price is only used in the remove liquidity process.
Status	Resolved The Alpaca Finance team has resolved this issue by using the maximum price for the profit and loss of short positions calculation in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.5.1. Description

Normally, when the trader closes their short position, the platform always calculates the profit and loss from the current maximum price to minimize the trader profit by using price from the `getMaxPrice()` function as shown in the `getDelta()` function at line 282.

GetterFacet.sol

```
252 function getDelta(
253   address indexToken,
254   uint256 size,
255   uint256 averagePrice,
256   bool isLong,
257   uint256 lastIncreasedTime,
258   int256 entryFundingRate,
259   int256 fundingFeeDebt
260 )
261   public
262   view
263   returns (
264     bool,
265     uint256,
    int256
  )
{
  GetDeltaLocalVars memory vars;

  // Load diamond storage
  LibPoolV1.PoolV1DiamondStorage storage ds = LibPoolV1
    .poolV1DiamondStorage();

  // Load PoolConfigV1 diamond storage
  LibPoolConfigV1.PoolConfigV1DiamondStorage
    storage poolConfigDs = LibPoolConfigV1.poolConfigV1DiamondStorage();

  if (averagePrice == 0) revert GetterFacet_InvalidAveragePrice();
  vars.price = isLong
    ? ds.oracle.getMinPrice(indexToken)
    : ds.oracle.getMaxPrice(indexToken);

  unchecked {
    vars.priceDelta = averagePrice > vars.price
      ? averagePrice - vars.price
      : vars.price - averagePrice;
  }
  vars.delta = int256((size * vars.priceDelta) / averagePrice);

  if (isLong) {
    vars.delta = vars.price > averagePrice ? vars.delta : -vars.delta;
  } else {
    vars.delta = vars.price < averagePrice ? vars.delta : -vars.delta;
  }

  // Negative funding fee means profit to the position
  vars.fundingFee =
    getFundingFee(indexToken, isLong, size, entryFundingRate) +
    fundingFeeDebt;
  vars.delta -= vars.fundingFee;
  vars.isProfit = vars.delta > 0;
  vars.unsignedDelta = vars.delta > 0
    ? uint256(vars.delta)
    : uint256(-vars.delta);

  vars.minBps = block.timestamp >
    lastIncreasedTime + poolConfigDs.minProfitDuration
    ? 0
    : poolConfigDs.tokenMetas[indexToken].minProfitBps;
  if (vars.isProfit && vars.unsignedDelta * BPS <= size * vars.minBps)
    vars.unsignedDelta = 0;
```

```
313     return (vars.isProfit, vars.unsignedDelta, vars.fundingFee);
314   }
```

On the other hand, in the `removeLiquidity()` of the `LiquidityFacet` contract, the minimum price is used to calculate the asset under management value, as shown in line 293 through the `getAumE18()` function, which will call the `getAum()` function internally.

LiquidityFacet.sol

```
276   function removeLiquidity(
277     address account,
278     address tokenOut,
279     address receiver
280   ) external nonReentrant allowed(account) returns (uint256) {
281     // LOAD diamond storage
282     LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
283       .poolV1DiamondStorage();
284
285     uint256 liquidity = poolV1ds.alp.balanceOf(address(this));
286
287     if (!LibPoolConfigV1.isAcceptToken(tokenOut))
288       revert LiquidityFacet_BadToken();
289     if (liquidity == 0) revert LiquidityFacet_BadAmount();
290
291     LibPoolV1.realizedFarmPnL(tokenOut);
292
293     uint256 aum = GetterFacetInterface(address(this)).getAumE18(false);
294     uint256 lpSupply = poolV1ds.alp.totalSupply();
295
296     uint256 lpUsdValue = (liquidity * aum) / lpSupply;
297     // Adjust totalUsdDebt if lpUsdValue > totalUsdDebt.
298     if (poolV1ds.totalUsdDebt < lpUsdValue)
299       poolV1ds.totalUsdDebt += lpUsdValue - poolV1ds.totalUsdDebt;
300     uint256 amountOut = _exit(
301       tokenOut,
302       lpUsdValue,
303       receiver,
304       account,
305       LiquidityAction.REMOVE_LIQUIDITY
306     );
307
308     poolV1ds.alp.burn(address(this), liquidity);
309     LibPoolV1.tokenOut(tokenOut, receiver, amountOut);
310
311     emit RemoveLiquidity(
312       account,
313       tokenOut,
    liquidity,
    aum,
    lpSupply,
    lpUsdValue,
    amountOut
  );

  return amountOut;
}
```

In the getAum() function, the calculation also includes an unrealized profit and loss calculation of total short positions on the platform in lines 737 - 755.

GetterFacet.sol

```
712 function getAum(bool isUseMaxPrice) public view returns (uint256) {
713   LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
714     .poolV1DiamondStorage();
715
716   address token = LibPoolConfigV1.getNextAllowTokenOf(LINKEDLIST_START);
717   uint256 aum = poolV1ds.additionalAum;
718   uint256 shortProfits = 0;
719
720   while (token != LINKEDLIST_END) {
721     uint256 price = !isUseMaxPrice
722       ? poolV1ds.oracle.getMinPrice(token)
723       : poolV1ds.oracle.getMaxPrice(token);
724     uint256 liquidity = poolV1ds.liquidityOf[token];
725     uint256 decimals = LibPoolConfigV1.getTokenDecimalsOf(token);
726
727     // Handle strategy delta
728     (bool isStrategyProfit, uint256 strategyDelta) = LibPoolConfigV1
729       .getStrategyDelta(token);
730     if (isStrategyProfit) liquidity += strategyDelta;
731     else liquidity -= strategyDelta;
732
733     if (LibPoolConfigV1.isStableToken(token)) {
734       aum += (liquidity * price) / 10**decimals;
735     } else {
736       uint256 shortSize = poolV1ds.shortSizeOf[token];
737       if (shortSize > 0) {
738         uint256 shortAveragePrice = poolV1ds.shortAveragePriceOf[token];
739         uint256 priceDelta;
740         unchecked {
741           priceDelta = shortAveragePrice > price
742             ? shortAveragePrice - price
743             : price - shortAveragePrice;
744         }
```

```
        // Findout delta (can be either profit or loss) of short positions.
        uint256 delta = (shortSize * priceDelta) / shortAveragePrice;

        if (price > shortAveragePrice) {
          // Short position is at loss, then count it as aum
          aum += delta;
        } else {
          // Short position is at profit, then count it as shortProfits
          shortProfits += delta;
        }
      }

      // Add guaranteed USD to the aum.
      aum += poolV1ds.guaranteedUsdOf[token];

      // Add actual liquidity of the token to the aum.
      aum +=
        ((liquidity - poolV1ds.reservedOf[token]) * price) /
        10**decimals;
    }

    token = LibPoolConfigV1.getNextAllowTokenOf(token);
  }
  aum = shortProfits > aum ? 0 : aum - shortProfits;
  return
    poolV1ds.discountedAum > aum
      ? 0
      : aum -
        poolV1ds.discountedAum -
        poolV1ds.fundingFeePayable +
        poolV1ds.fundingFeeReceivable;
}
```

However, when the `getAum()` function is called with `isUseMaxPrice = false` for example in the `removeLiquidity()` function, the profit and loss of short positions will be calculated from the minimum price which result in slightly lesser than the actual value.

5.5.2. Remediation

Inspex suggests using the maximum price for the profit and loss of short positions calculation.

GetterFacet.sol

```
function getAum(bool isUseMaxPrice) public view returns (uint256) {
  LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
    .poolV1DiamondStorage();

  address token = LibPoolConfigV1.getNextAllowTokenOf(LINKEDLIST_START);
```

```
uint256 aum = poolV1ds.additionalAum;
uint256 shortProfits = 0;

while (token != LINKEDLIST_END) {
  uint256 price = !isUseMaxPrice
    ? poolV1ds.oracle.getMinPrice(token)
    : poolV1ds.oracle.getMaxPrice(token);
  uint256 liquidity = poolV1ds.liquidityOf[token];
  uint256 decimals = LibPoolConfigV1.getTokenDecimalsOf(token);

  // Handle strategy delta
  (bool isStrategyProfit, uint256 strategyDelta) = LibPoolConfigV1
    .getStrategyDelta(token);
  if (isStrategyProfit) liquidity += strategyDelta;
  else liquidity -= strategyDelta;

  if (LibPoolConfigV1.isStableToken(token)) {
    aum += (liquidity * price) / 10**decimals;
  } else {
    uint256 shortSize = poolV1ds.shortSizeOf[token];
    if (shortSize > 0) {
      uint256 shortAveragePrice = poolV1ds.shortAveragePriceOf[token];
      uint256 priceDelta;
      uint256 maxPrice = poolV1ds.oracle.getMaxPrice(token);
      unchecked {
        priceDelta = shortAveragePrice > maxPrice
          ? shortAveragePrice - maxPrice
          : maxPrice - shortAveragePrice;
      }
      // Findout delta (can be either profit or loss) of short positions.
      uint256 delta = (shortSize * priceDelta) / shortAveragePrice;

      if (maxPrice > shortAveragePrice) {
        // Short position is at loss, then count it as aum
        aum += delta;
      } else {
        // Short position is at profit, then count it as shortProfits
        shortProfits += delta;
      }
    }

    // Add guaranteed USD to the aum.
    aum += poolV1ds.guaranteedUsdOf[token];

    // Add actual liquidity of the token to the aum.
    aum +=
      ((liquidity - poolV1ds.reservedOf[token]) * price) /
```

```
        10**decimals;
    }

    token = LibPoolConfigV1.getNextAllowTokenOf(token);
  }
  aum = shortProfits > aum ? 0 : aum - shortProfits;
  return
    poolV1ds.discountedAum > aum
      ? 0
      : aum -
        poolV1ds.discountedAum -
        poolV1ds.fundingFeePayable +
        poolV1ds.fundingFeeReceivable;
}
```

5.6. Denial of Service in PoolRouter03

ID	IDX-006
Target	PoolRouter03
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Low The user is unable to execute some of the functions that transfer a native token after paying a fee. However, the user can use alternative functions that use a wrap token instead. **Likelihood:** High This will occur every time when calling the functions that transfer a native token after paying a fee in the `PoolRouter03` contract.
Status	Resolved The Alpaca Finance team has resolved this issue by calculating the deposit value properly in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.6.1. Description

In the `PoolRouter03` contract, the user may need to pay a fee by the `msg.value` when calling the functions in the contract via the `_updatePrices` function, as shown below in line 65.

PoolRouter03.sol

```
60  function _updatePrices(bytes[] memory _priceUpdateData) internal {
61    uint256 fee = oraclePriceUpdater.getUpdateFee(_priceUpdateData);
62    if (fee == 0) return;
63    if (fee > msg.value)
64      revert PoolRouter_InsufficientUpdatedFee(fee, msg.value);
65    oraclePriceUpdater.updatePrices{ value: fee }(_priceUpdateData);
66  }
```

However, when calling the function that also uses the `msg.value`, if the user already paid a fee in the `_updatePrices()` function, the transferring of the `msg.value` after that will be reverted since the `msg.value` has been decremented and the token will not be enough for the `msg.value` in line 128.

PoolRouter03.sol

```
121  function addLiquidityNative(
122    address token,
123    address receiver,
  uint256 minLiquidity,
  bytes[] memory _priceUpdateData
) external payable returns (uint256) {
  _updatePrices(_priceUpdateData);
  WNATIVE.deposit{ value: msg.value }();
  IERC20(address(WNATIVE)).safeTransfer(address(pool), msg.value);

  uint256 receivedAmount = LiquidityFacetInterface(pool).addLiquidity(
    msg.sender,
    token,
    receiver
  );

  if (receivedAmount < minLiquidity)
    revert PoolRouter_InsufficientOutputAmount(minLiquidity, receivedAmount);

  return receivedAmount;
}
```

The functions that transfer using `msg.value` after paying a fee via the `_updatePrices()` function are as follows:

File	Contract	Function
PoolRouter03.sol (L:121)	PoolRouter03	addLiquidityNative()
PoolRouter03.sol (L:234)	PoolRouter03	increasePositionNative()
PoolRouter03.sol (L:406)	PoolRouter03	swapNative()

5.6.2. Remediation

Inspex suggests modifying the function to use the local variable, which is reduced by a fee for transferring after that, for example:

PoolRouter03.sol

```
function _updatePrices(bytes[] memory _priceUpdateData) internal
returns(uint256) {
  uint256 fee = oraclePriceUpdater.getUpdateFee(_priceUpdateData);
  if (fee == 0) return;
  if (fee >= msg.value)
    revert PoolRouter_InsufficientUpdatedFee(fee, msg.value);
  oraclePriceUpdater.updatePrices{ value: fee }(_priceUpdateData);
  return fee;
}
```

PoolRouter03.sol

```
function addLiquidityNative(
  address token,
  address receiver,
  uint256 minLiquidity,
  bytes[] memory _priceUpdateData
) external payable returns (uint256) {
  uint256 fee = _updatePrices(_priceUpdateData);
  uint256 amountIn = msg.value - fee;
  WNATIVE.deposit{ value: amountIn }();
  IERC20(address(WNATIVE)).safeTransfer(address(pool), amountIn);

  uint256 receivedAmount = LiquidityFacetInterface(pool).addLiquidity(
    msg.sender,
    token,
    receiver
  );

  if (receivedAmount < minLiquidity)
    revert PoolRouter_InsufficientOutputAmount(minLiquidity, receivedAmount);

  return receivedAmount;
}
```

5.7. Improper Share Calculation

ID	IDX-007
Target	ALPStaking
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The number of tokens is used to calculate shares rather than their value, which implies that high-value tokens receive a smaller share than they should. This results in the available rewards for platform users being incorrect. **Likelihood:** Medium Only the contract owner could add allowed staking tokens into the contract. The share calculation will be accurate if the owner adds only the $ALP and not any other tokens.
Status	Resolved The Alpaca Finance team has resolved this issue by removing the multi staking token mechanism and only allowing $ALP for staking in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.7.1. Description

The `ALPStaking` contract is responsible for providing the share amount of the users' staked tokens of the platform. By design, the contract can handle multiple staking tokens in one pool.

The `calculateShare()` function calculates the share of a user by summing the value of the `userTokenAmount` state for each staking token held by the user. It is important to note that the `userTokenAmount` state represents the amount of the staking token, not the value of the staking token itself. In the other words, all staking tokens will have the same influence on the final share amount, regardless of their value.

ALPStaking.sol

```
function calculateShare(address rewarder, address user)
    external
    view
    returns (uint256)
{
    address[] memory tokens = rewarderStakingTokens[rewarder];
    uint256 share = 0;
    uint256 length = tokens.length;
    for (uint256 i = 0; i < length; ) {
        share += userTokenAmount[tokens[i]][user];

        unchecked {
            ++i;
        }
    }
    return share;
}
```

Similar to the `calculateShare()` function, the `calculateTotalShare()` function calculates the total share of the pool. However, instead of using the `userTokenAmount` state, it uses the `balanceOf(address(this))` value to calculate the `totalShare` value. This approach also treats all staking tokens having the same value.

ALPStaking.sol

```
function calculateTotalShare(address rewarder)
    external
    view
    returns (uint256)
{
    address[] memory tokens = rewarderStakingTokens[rewarder];
    uint256 totalShare = 0;
    uint256 length = tokens.length;
    for (uint256 i = 0; i < length; ) {
        totalShare += IERC20Upgradeable(tokens[i]).balanceOf(address(this));

        unchecked {
            ++i;
        }
    }
    return totalShare;
}
```

5.7.2. Remediation

In the case that the `ALPStaking` contract only allows users to stake the `$ALP` in the contract, Inspex suggests validating the token address in the `addStakingToken()` and `addRewarder()` functions to ensure that only the `$ALP` can be added. If multiple token staking is needed, the share calculation must be separate for each pool to calculate the proper reward value.

5.8. Design Flaw in Fixed Rate Token Swap

ID	IDX-008
Target	PoolOracle
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** High The attacker could perform an arbitrage swap between the DEX and Alpaca Finance's liquidity pool to gain the profit from the fixed rate swap. All profitable assets in the pool will be swapped out. **Likelihood:** Low The arbitrage opportunity occurs when the difference between the price on DEX and the platform's oracle is large enough for an arbitrager to be able to leverage for the profit.
Status	Resolved * The Alpaca Finance team has mitigated this issue by implementing the various security measures such as usage of price oracle of Chainlink and Pyth Network, tax fee charging as virtual price impact depending on the ALP token weights and 15-minutes liquidity cooldown on ALP deposit/withdrawal. These security measures could prevent attackers from exploiting this feature.

5.8.1. Description

The platform uses an oracle as a reference for the price of each asset in $USD. When a user performs a swap from the `LiquidityFacet` contract, the price of the swapped tokens will only depend on the prices from the Oracle of the platform in lines 413 and 414.

Therefore, if the price from the oracle deviates from the market price, there will be an opportunity for arbitraging on the platform. Then, the price-deviated token will fluctuate into the platform to drain another token from the platform, which causes an acute shortage of liquidity and a loss of the total value of the pool.

LiquidityFacet.sol

```
377  function swap(
378    address account,
379    address tokenIn,
380    address tokenOut,
381    uint256 minAmountOut,
382    address receiver
383  ) external nonReentrant returns (uint256) {
384    SwapLocalVars memory vars;
```

```

// LOAD diamond storage
LibPoolV1.PoolV1DiamondStorage storage poolV1ds = LibPoolV1
  .poolV1DiamondStorage();

// Pull Tokens
uint256 amountIn = LibPoolV1.pullTokens(tokenIn);

if (!LibPoolConfigV1.isSwapEnable()) revert LiquidityFacet_SwapDisabled();
if (!LibPoolConfigV1.isAcceptToken(tokenIn))
  revert LiquidityFacet_BadTokenIn();
if (!LibPoolConfigV1.isAcceptToken(tokenOut))
  revert LiquidityFacet_BadTokenOut();
if (tokenIn == tokenOut) revert LiquidityFacet_SameTokenInTokenOut();
if (amountIn == 0) revert LiquidityFacet_BadAmount();

LibPoolV1.realizedFarmPnL(tokenIn);
LibPoolV1.realizedFarmPnL(tokenOut);

FundingRateFacetInterface(address(this)).updateFundingRate(
  tokenIn,
  tokenIn
);
FundingRateFacetInterface(address(this)).updateFundingRate(
  tokenOut,
  tokenOut
);

vars.priceIn = poolV1ds.oracle.getMinPrice(tokenIn);
vars.priceOut = poolV1ds.oracle.getMaxPrice(tokenOut);

vars.amountOut = (amountIn * vars.priceIn) / vars.priceOut;
vars.amountOut = LibPoolV1.convertTokenDecimals(
  LibPoolConfigV1.getTokenDecimalsOf(tokenIn),
  LibPoolConfigV1.getTokenDecimalsOf(tokenOut),
  vars.amountOut
);

// Adjust USD debt as swap shifted the debt between two assets
vars.usdDebt = (amountIn * vars.priceIn) / PRICE_PRECISION;
vars.usdDebt = LibPoolV1.convertTokenDecimals(
  LibPoolConfigV1.getTokenDecimalsOf(tokenIn),
  USD_DECIMALS,
  vars.usdDebt
);

uint256 swapFeeBps = GetterFacetInterface(address(this)).getSwapFeeBps(
    tokenIn,
    tokenOut,
    vars.usdDebt
  );
  uint256 amountOutAfterFee = _collectSwapFee(
    tokenOut,
    poolV1ds.oracle.getMinPrice(tokenOut),
    vars.amountOut,
    swapFeeBps,
    account,
    LiquidityAction.SWAP
  );

  LibPoolV1.increasePoolLiquidity(tokenIn, amountIn);
  LibPoolV1.increaseUsdDebt(tokenIn, vars.usdDebt);

  LibPoolV1.decreasePoolLiquidity(tokenOut, vars.amountOut);
  LibPoolV1.decreaseUsdDebt(tokenOut, vars.usdDebt);

  // Buffer check
  if (
    poolV1ds.liquidityOf[tokenOut] <
    LibPoolConfigV1.getTokenBufferLiquidityOf(tokenOut)
  ) revert LiquidityFacet_LiquidityBuffer();

  // Slippage check
  if (amountOutAfterFee < minAmountOut) revert LiquidityFacet_Slippage();

  // Transfer amount out.
  LibPoolV1.tokenOut(tokenOut, receiver, amountOutAfterFee);
  emit Swap(
    receiver,
    tokenIn,
    tokenOut,
    amountIn,
    vars.amountOut,
    amountOutAfterFee,
    swapFeeBps
  );

  return amountOutAfterFee;
}
```

5.8.2. Remediation

Inspex suggests implementing the mechanism to prevent the arbitrager from profitable action against the liquidity pool, such as applying the cooldown on the add/remove liquidity, allowing only the platform itself to perform the swap at the fixed price rate.

5.9. Inconsistent swap() function in RewardDistributor

ID	IDX-009
Target	RewardDistributor
Category	Advanced Smart Contract Vulnerability
CWE	CWE-1164: Irrelevant Code
Risk	**Severity:** Medium **Impact:** Low The platform fee is unable to be distributed due to a lack of the `PoolRouter03` contract support. However, the platform fee can still be distributed normally when the `RewardDistributor` has been replaced by the platform owner. **Likelihood:** High This issue occurs every time the reward is being distributed and the `RewardDistributor` contract is trying to swap with the `PoolRouter03` contract.
Status	Resolved The Alpaca Finance team has resolved this issue by implementing the `PoolRouter03.swap()` function instead to properly update the price feed before swapping in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.9.1. Description

In the `RewardDistributor` contract, the `claimAndFeedProtocolRevenue()` is called by the feeder in order to withdraw the fee, swap to reward token and distribute it to the rewarder.

RewardDistributor.sol

```
function claimAndFeedProtocolRevenue(
  address[] memory tokens,
  uint256 feedingExpiredAt,
  uint256 weekTimestamp,
  uint256 referralRevenueAmount,
  bytes32 merkleRoot
) external onlyFeeder {
  _claimAndSwap(tokens);
  _feedProtocolRevenue(
    feedingExpiredAt,
    weekTimestamp,
    referralRevenueAmount,
    merkleRoot
  );
}
```

The `_claimAndSwap()` function withdraws fee then swaps it with the `_swapTokenToRewardToken()` function.

RewardDistributor.sol

```
function _claimAndSwap(address[] memory tokens) internal {
  uint256 length = tokens.length;
  for (uint256 i = 0; i < length; ) {
    // 1. Withdraw protocol revenue
    _withdrawProtocolRevenue(tokens[i]);
    // 2. Swap those revenue (along with surplus) to RewardToken Token
    _swapTokenToRewardToken(
      tokens[i],
      IERC20Upgradeable(tokens[i]).balanceOf(address(this))
    );

    unchecked {
      i++;
    }
  }
}
```

However, the swap function call to the `poolRouter` contract in line 369 is `IPoolRouter.swap()` function, which is inconsistent with the `PoolRouter03.swap()` function which requires `_priceUpdateData` parameter.

RewardDistributor.sol

```
358 function _swapTokenToRewardToken(address token, uint256 amount) internal {
359   // If no token, no need to swap
360   if (amount == 0) return;
361
362   // If token is already reward token, no need to swap
363   if (token == rewardToken) return;
364
365   // Approve the token
366   IERC20Upgradeable(token).approve(poolRouter, amount);
367
368   // Swap
369   IPoolRouter(poolRouter).swap(token, rewardToken, amount, 0, address(this));
370 }
```

PoolRouter03.sol

```
function swap(
  address tokenIn,
  address tokenOut,
  uint256 amountIn,
  uint256 minAmountOut,
  address receiver,
  bytes[] memory _priceUpdateData
) external payable returns (uint256) {
  _updatePrices(_priceUpdateData);
  return
    _swap(msg.sender, tokenIn, tokenOut, amountIn, minAmountOut, receiver);
}
```

5.9.2. Remediation

Inspex suggests implementing the `PoolRouter03.swap()` function instead to properly update the price feed before swapping, for example:

RewardDistributor.sol

```
function _swapTokenToRewardToken(
  address token,
  uint256 amount,
  bytes[] memory priceUpdateData,
  uint256 fee
) internal {
  // If no token, no need to swap
  if (amount == 0) return;

  // If token is already reward token, no need to swap
  if (token == rewardToken) return;

  // Approve the token
  IERC20Upgradeable(token).approve(poolRouter, amount);

  // Swap
  IPoolRouter(poolRouter).swap{ value: fee }(
    token,
    rewardToken,
    amount,
    0,
    address(this),
    priceUpdateData
  );
}
```

5.10. Unsafe Token Transfer in RewardDistributor

ID	IDX-010
Target	RewardDistributor
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standard
Risk	**Severity:** Low **Impact:** Medium The users may lose their reward tokens due to improperly implemented ERC20 tokens. **Likelihood:** Low Only improperly implemented token that does not revert the transaction on the invalid transfer amount is affected. Furthermore, only the contract owner can add the `rewardToken`.
Status	Resolved The Alpaca Finance team has resolved this issue by replacing the `transfer()` function with `safeTransfer()` function from OpenZeppelin's SafeERC20 contract in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.10.1. Description

External ERC20 tokens can be added to the contract as the `rewardToken` using the `initialize()` and `setParams()` functions. ERC20 tokens can be improperly implemented, allowing the execution of failed `transfer()` function without reverting when the invalid transfer amount occurs. This can cause significant damage to the smart contract if not enough tokens are available.

In the `RewardDistributor` contract, the `_feedProtocolRevenue()` function is used to feed the `rewardToken` for protocol revenues at lines 273, 276 and 279.

RewardDistributor.sol

```
240 function _feedProtocolRevenue(
241   uint256 feedingExpiredAt,
242   uint256 weekTimestamp,
243   uint256 referralRevenueAmount,
244   bytes32 merkleRoot
245 ) internal {
246   // Transfer referral revenue to merkle airdrop address for distribution
247   uint256 totalProtocolRevenue = IERC20Upgradeable(rewardToken).balanceOf(
248     address(this)
249   );
250   // totalProtocolRevenue * referralRevenueMaxThreshold / 10000 <
```

```
    referralRevenueAmount
251   if (
252     totalProtocolRevenue * referralRevenueMaxThreshold <
253     referralRevenueAmount * MAX_BPS
254   ) revert RewardDistributor_ReferralRevenueExceedMaxThreshold();
255   merkleAirdrop.init(weekTimestamp, merkleRoot);
256   IERC20Upgradeable(rewardToken).safeTransfer(
257     address(merkleAirdrop),
258     referralRevenueAmount
259   );
260
261   // Calculate reward sharing
262   (
263     uint256 alpStakingRewardAmount,
264     uint256 devFundAmount,
265     uint256 govRewardAmount,
266     uint256 burnAMount
267   ) = _calculateRewardSharing();
268
269   // Feed for protocol revenue.
270   _feedRewardToRewarders(feedingExpiredAt, alpStakingRewardAmount);
271
272   // Collect Dev Fund
273   _collectDevFund(rewardToken, devFundAmount);
274
275   // Collect Gov Reward
276   _collectGovReward(rewardToken, govRewardAmount);
277
278   // Collect Gov Reward
279   _collectBurn(rewardToken, burnAMount);
280
281   emit LogProtocolFee(
282     weekTimestamp,
283     referralRevenueAmount,
284     alpStakingRewardAmount,
285     devFundAmount,
286     govRewardAmount,
287     burnAMount
288   );
289 }
```

The `_collectDevFund()`, `_collectGovReward()` and `_collectBurn()` functions are used to transfer the reward sharing for `devFundAddress`, `govFeeder` and `burner` at lines 341, 348 and 355, respectively.

RewardDistributor.sol

```
337 function _collectDevFund(address _token, uint256 _amount) internal {
338   // If no token, no need transfer
339     if (_amount == 0) return;
340 
341     IERC20Upgradeable(_token).transfer(devFundAddress, _amount);
342 }
343 
344 function _collectGovReward(address _token, uint256 _amount) internal {
345     // If no token, no need transfer
346     if (_amount == 0) return;
347 
348     IERC20Upgradeable(_token).transfer(govFeeder, _amount);
349 }
350 
351 function _collectBurn(address _token, uint256 _amount) internal {
352     // If no token, no need transfer
353     if (_amount == 0) return;
354 
355     IERC20Upgradeable(_token).transfer(burner, _amount);
356 }
```

But, if the `rewardToken` is improperly implemented, such as by missing the return value while transferring tokens. The `transfer()` function calls above will fail, and the feed protocol revenue transaction will always revert. Resulting in the reward revenue being permanently stuck in the contract.

5.10.2. Remediation

Inspex suggests replacing the `transfer()` function of the `_token` with the `safeTransfer()` function from OpenZeppelin's SafeERC20 contract at line 341, 348 and 355, for example:

RewardDistributor.sol

```
337 function _collectDevFund(address _token, uint256 _amount) internal {
338     // If no token, no need transfer
339     if (_amount == 0) return;
340 
341     IERC20Upgradeable(_token).safeTransfer(devFundAddress, _amount);
342 }
343 
344 function _collectGovReward(address _token, uint256 _amount) internal {
345     // If no token, no need transfer
346     if (_amount == 0) return;
347 
348     IERC20Upgradeable(_token).safeTransfer(govFeeder, _amount);
349 }
350 
351 function _collectBurn(address _token, uint256 _amount) internal {
352     // If no token, no need transfer
353     if (_amount == 0) return;
354 
```

```
    IERC20Upgradeable(_token).safeTransfer(burner, _amount);
}
```

5.11. Oracle Price Update Bypass

ID	IDX-011
Target	PoolOracle PythPriceFeed
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Medium The secondary price oracle's (Pyth) update logic does not enforce; the attacker can select prices sources while interacting with the platform. This results in an unfair advantage to the platform users. **Likelihood:** Low This issue only occurs when the prices from the primary and secondary price oracles are different, which depends on how the `maxPriceAge` state is set by the platform.
Status	Resolved The Alpaca Finance team has resolved this issue by decreasing the `MAXIMUM_PRICE_AGE` constant value and setting the maxPriceAge state to 2 minutes in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.11.1. Description

In general, the `_priceUpdateData` parameter is supplied by the user when they interact via `PoolRouter03` or `OrderBook02` contract. If the supplied value is an empty array, or old data, the price will not be updated. Hence, the manual price update from the user can be skipped.

PoolRouter03

```
function addLiquidity(
  address token,
  uint256 amount,
  address receiver,
  uint256 minLiquidity,
  bytes[] memory _priceUpdateData
) external payable returns (uint256) {
  _updatePrices(_priceUpdateData);
  IERC20(token).safeTransferFrom(msg.sender, address(pool), amount);

  uint256 receivedAmount = LiquidityFacetInterface(pool).addLiquidity(
    msg.sender,
    token,
    receiver
  );

  if (receivedAmount < minLiquidity)
    revert PoolRouter_InsufficientOutputAmount(minLiquidity, receivedAmount);

  return receivedAmount;
}
```

In the PoolOracle contract, the _getPrice() function is used to get the price from the price oracle, depending on the isSecondaryPriceEnabled state.

PoolOracle.sol

```
function _getPrice(address token, bool isUseMaxPrice)
  internal
  view
  returns (uint256)
{
  uint256 price = _getPrimaryPrice(token, isUseMaxPrice);

  if (isSecondaryPriceEnabled) {
    price = getSecondaryPrice(token, price, isUseMaxPrice);
  }

  // Handle strict stable price deviation.
  // SLOAD
  PriceFeedInfo memory priceFeed = priceFeedInfo[token];
  if (address(priceFeed.priceFeed) == address(0))
    revert PoolOracle_PriceFeedNotAvailable();
  if (priceFeed.isStrictStable) {
    uint256 delta;
    unchecked {
      delta = price > ONE_USD ? price - ONE_USD : ONE_USD - price;
    }

    if (delta <= maxStrictPriceDeviation) return ONE_USD;

    if (isUseMaxPrice && price > ONE_USD) return price;

    if (!isUseMaxPrice && price < ONE_USD) return price;

    return ONE_USD;
  }

  // Handle spreadBasisPoint
  if (isUseMaxPrice) return (price * (BPS + priceFeed.spreadBps)) / BPS;

  return (price * (BPS - priceFeed.spreadBps)) / BPS;
}
```

At line 86, the `getSecondaryPrice()` function will be called to get the price from the Pyth oracle while passing the price from Chainlink oracle as the `_referencePrice` parameter.

PoolOracle.sol

```
function getSecondaryPrice(
  address _token,
  uint256 _referencePrice,
  bool _maximise
) public view returns (uint256) {
  if (secondaryPriceFeed == address(0)) {
    return _referencePrice;
  }
  return
    ISecondaryPriceFeed(secondaryPriceFeed).getPrice(
      _token,
      _referencePrice,
      _maximise
    );
}
```

The `getPrice` function in the `PythPriceFeed` contract is used to get the price from the Pyth oracle.

PythPriceFeed.sol

```
function getPrice(
  address _token,
  uint256 _referencePrice,
  bool _maximise
) external view returns (uint256) {
  if (favorRefPrice) {
    return _referencePrice;
  }

  bytes32 priceID = tokenPriceId[_token];
  // Read the current value of priceID, aborting the transaction if the price has not been updated recently.
  // Every chain has a default recency threshold which can be retrieved by calling the getValidTimePeriod() function on the contract.
  // Please see IPyth.sol for variants of this function that support configurable recency thresholds and other useful features.

  try pyth.getPriceNoOlderThan(priceID, maxPriceAge) returns (
    PythStructs.Price memory _price
  ) {
```

```
  uint256 tokenDecimals = _price.expo < 0
   ? (10**int256(-_price.expo).toUint256())
   : 10**int256(_price.expo).toUint256();
  if (_maximise) {
   return
    ((int256(_price.price) + uint256(_price.conf).toInt256())
     .toUint256() * PRICE_PRECISION) / tokenDecimals;
  }
  return
   ((int256(_price.price) - uint256(_price.conf).toInt256()).toUint256() *
    PRICE_PRECISION) / tokenDecimals;
 } catch {
  // if some problem occurred (e.g. price is older than maxPriceAge), return reference price from primary source
  return _referencePrice;
 }
}
```

By making an external call to the `pyth.getPriceNoOlderThan()` function, it can be reverted if the price age is greater than the `maxPriceAge` state, in which case the `_referencePrice` will be used as the return value for `getPrice()` function.

The following `setMaxPriceAge()` function shows that the `maxPriceAge` state must be set below MAXIMUM_PRICE_AGE constant.

PythPriceFeed.sol

```
uint256 public constant MAXIMUM_PRICE_AGE = 3600; // 1 hour
```

PythPriceFeed.sol

```
function setMaxPriceAge(uint256 _maxPriceAge) external onlyOwner {
 if (_maxPriceAge > MAXIMUM_PRICE_AGE) {
  revert PythPriceFeed_InvalidMaxPriceAge();
 }
 maxPriceAge = _maxPriceAge;

 emit SetMaxPriceAge(_maxPriceAge);
}
```

This results in the user being able to choose the favorable price by skipping the updated price or not when the `maxPriceAge` state is wide enough to be profitable, such as half an hour.

5.11.2. Remediation

Inspex suggests decreasing the `MAXIMUM_PRICE_AGE` constant value and setting the `maxPriceAge` state as low as possible (maximum 2 minutes, same as Chainlink) to prevent the price gap between two price oracles.

PythPriceFeed.sol

```solidity
uint256 public constant MAXIMUM_PRICE_AGE = 120; // 2 mins
```

If modifying the `MAXIMUM_PRICE_AGE` constant value is not possible, Inspex suggests mitigating the risk of this issue by using a schedule for an update on the price of the Pyth Oracle, e.g. every 2 minutes.

5.12. Missing Input Validation in bulkClaim() Function

ID	IDX-012
Target	MerkleAirdrop
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Very Low **Impact:** Low The `bulkClaim()` function can be reverted due to the array being out of bounds in looping. This is because there is no input validation for input parameters. **Likelihood:** Low It is unlikely that the execution of `bulkClaim()` function will fail since the input parameters are provided by the caller. Moreover, the `claim()` function can be called on an individual claim.
Status	Resolved The Alpaca Finance team has resolved this issue by adding an input validation length check for input parameters in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.12.1. Description

The `bulkClaim()` function in the `MerkleAirdrop` contract allows the user to claim the airdrop.

MerkleAirdrop.sol

```
function bulkClaim(
  uint256[] calldata weekTimestamps,
  uint256[] calldata indices,
  address[] calldata accounts,
  uint256[] calldata amounts,
  bytes32[][] calldata merkleProof
) external {
  for (uint256 i = 0; i < weekTimestamps.length; ) {
    _claim(
      weekTimestamps[i],
      indices[i],
      accounts[i],
      amounts[i],
      merkleProof[i]
    );
    unchecked {
      i++;
    }
  }
}
```

However, there is no array length check for input parameters. This results in execution failing when calling the function with an unmatched array length.

5.12.2. Remediation

Inspex suggests adding an input validation length check for input parameters, for example:

MerkleAirdrop.sol

```
function bulkClaim(
  uint256[] calldata weekTimestamps,
  uint256[] calldata indices,
  address[] calldata accounts,
  uint256[] calldata amounts,
  bytes32[][] calldata merkleProof
) external {
  if (
    weekTimestamps.length != indices.length ||
    weekTimestamps.length != accounts.length ||
    weekTimestamps.length != amounts.length ||
    weekTimestamps.length != merkleProof.length
  )
    revert MerkleAirdrop_InvalidLength();
  for (uint256 i = 0; i < weekTimestamps.length; ) {
    _claim(
      weekTimestamps[i],
      indices[i],
      accounts[i],
      amounts[i],
      merkleProof[i]
    );
    unchecked {
      i++;
    }
  }
}
```

5.13. Insufficient Logging for Privileged Functions

ID	IDX-013
Target	MerkleAirdrop WNativeRelayer
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved The Alpaca Finance team has resolved this issue by emitting events for the execution of privileged functions in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.13.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the `onlyOwner` role can call the `emergencyWithdraw()` function to transfer all tokens to the `receiver` address at line 154, and no events are emitted.

MerkleAirdrop.sol

```
151  function emergencyWithdraw(address receiver) external onlyOwner {
152    IERC20 tokenContract = IERC20(token);
153    uint256 balance = tokenContract.balanceOf(address(this));
154    tokenContract.safeTransfer(receiver, balance);
155  }
```

The privileged functions without sufficient logging are as follows:

File	Contract	Function	Modifier
MerkleAirdrop.sol (L:151)	MerkleAirdrop	emergencyWithdraw()	onlyOwner
WNativeRelayer.sol (L:34)	WNativeRelayer	setCallerOk()	onlyOwner

5.13.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

MerkleAirdrop.sol

```
event EmergencyWithdraw(address _receiver, uint256 _balance);
function emergencyWithdraw(address receiver) external onlyOwner {
  IERC20 tokenContract = IERC20(token);
  uint256 balance = tokenContract.balanceOf(address(this));
  tokenContract.safeTransfer(receiver, balance);
  emit EmergencyWithdraw(receiver, balance);
}
```

5.14. Unnecessary Zero Amount Transfer

ID	IDX-014
Target	OrderBook02
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Alpaca Finance team has resolved this issue by modifying the validation by using < instead of <= when paying an executor fee in the execute order functions in commit `273f7a69df94d3031a8c811076440a3cfd99bf29`.

5.14.1. Description

In the `OrderBook02` contract, the `_transferOutETH()` function is used for transferring a native token from the contract.

OrderBook02.sol

```
function _transferOutETH(uint256 _amountOut, address _receiver) private {
    // prevent istanbul msg.sender.transfer problem
    IERC20Upgradeable(weth).safeTransfer(wnativeRelayer, _amountOut);
    IWNativeRelayer(wnativeRelayer).withdraw(_amountOut);

    payable(_receiver).transfer(_amountOut);
}
```

The `_transferOutETH` function is called inside the execute order functions for paying a fee for the `_feeReceiver`. For example, in line 575 of the `executeSwapOrder()` function:

OrderBook02.sol

```
530 function executeSwapOrder(
531   address _account,
532   uint256 _orderIndex,
533   address payable _feeReceiver,
534   bytes[] memory _priceUpdateData
535 ) external nonReentrant whitelisted {
536   uint256 updatePriceFee = _updatePrices(_priceUpdateData);
537   SwapOrder memory order = swapOrders[_account][_orderIndex];
```

```
  if (order.account == address(0)) revert NonExistentOrder();

  if (order.triggerAboveThreshold) {
    // gas optimisation
    // order.minAmount should prevent wrong price execution in case of simple limit order
    if (
      !validateSwapOrderPriceWithTriggerAboveThreshold(
        order.path,
        order.triggerRatio
      )
    ) revert InvalidPriceForExecution();
  }

  delete swapOrders[_account][_orderIndex];
  _removeFromOpenOrders(_account, 0, _orderIndex, OrderType.SWAP);

  IERC20Upgradeable(order.path[0]).safeTransfer(pool, order.amountIn);

  uint256 _amountOut;
  if (order.path[order.path.length - 1] == weth && order.shouldUnwrap) {
    _amountOut = _swap(
      order.account,
      order.path,
      order.minOut,
      address(this)
    );
    _transferOutETH(_amountOut, payable(order.account));
  } else {
    _amountOut = _swap(
      order.account,
      order.path,
      order.minOut,
      order.account
    );
  }
  if (updatePriceFee <= order.executionFee) {
    // pay executor
    _transferOutETH(order.executionFee - updatePriceFee, _feeReceiver);
  }

  emit ExecuteSwapOrder(
    _account,
    _orderIndex,
    order.path,
    order.amountIn,
    order.minOut,
584     order.triggerRatio,
585     order.triggerAboveThreshold,
586     order.shouldUnwrap,
587     order.executionFee,
588     _amountOut
589   );
590 }
```

However, the `_amountOut` can be `0` when the `updatePriceFee` is equal to the `order.executionFee`. None of the tokens will be transferred to the `_feeReceiver` address. This results in an unnecessary gas consuming operation.

The execute order functions that may transfer `0` tokens to the `_feeReceiver` address are as follows:

File	Contract	Function
OrderBook02.sol (L:530)	OrderBook02	executeSwapOrder()
OrderBook02.sol (L:869)	OrderBook02	executeIncreaseOrder()
OrderBook02.sol (L:1015)	OrderBook02	executeDecreaseOrder()

5.14.2. Remediation

Inspex suggests modifying the validation by using < instead of <= when paying an executor fee in the execute order functions, for example:

OrderBook02.sol

```
530 function executeSwapOrder(
531   address _account,
532   uint256 _orderIndex,
533   address payable _feeReceiver,
534   bytes[] memory _priceUpdateData
535 ) external nonReentrant whitelisted {
536   uint256 updatePriceFee = _updatePrices(_priceUpdateData);
537   SwapOrder memory order = swapOrders[_account][_orderIndex];
538   if (order.account == address(0)) revert NonExistentOrder();
539 
540   if (order.triggerAboveThreshold) {
541     // gas optimisation
542     // order.minAmount should prevent wrong price execution in case of simple
limit order
543     if (
544       !validateSwapOrderPriceWithTriggerAboveThreshold(
545         order.path,
546         order.triggerRatio
547       )
```

```
    ) revert InvalidPriceForExecution();
  }

  delete swapOrders[_account][_orderIndex];
  _removeFromOpenOrders(_account, 0, _orderIndex, OrderType.SWAP);

  IERC20Upgradeable(order.path[0]).safeTransfer(pool, order.amountIn);

  uint256 _amountOut;
  if (order.path[order.path.length - 1] == weth && order.shouldUnwrap) {
    _amountOut = _swap(
      order.account,
      order.path,
      order.minOut,
      address(this)
    );
    _transferOutETH(_amountOut, payable(order.account));
  } else {
    _amountOut = _swap(
      order.account,
      order.path,
      order.minOut,
      order.account
    );
  }
  if (updatePriceFee < order.executionFee) {
    // pay executor
    _transferOutETH(order.executionFee - updatePriceFee, _feeReceiver);
  }

  emit ExecuteSwapOrder(
    _account,
    _orderIndex,
    order.path,
    order.amountIn,
    order.minOut,
    order.triggerRatio,
    order.triggerAboveThreshold,
    order.shouldUnwrap,
    order.executionFee,
    _amountOut
  );
}
```

5.15. Unoptimized Invariant Calculation

ID	IDX-015
Target	OrderBook02
Category	Smart Contract Best Practice
CWE	CWE-1164: Irrelevant Code
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Alpaca Finance team has resolved this issue by replacing the expression with 1 literal in commit 273f7a69df94d3031a8c811076440a3cfd99bf29.

5.15.1. Description

The validateSwapOrderPriceWithTriggerAboveThreshold() function in the OrderBook02 contract calculates and validates the price ratio of the swapping paths. The number of a legit swapping path in this platform is two and three.

The function will handle the paths with lengths two and three separately by using the if condition at line 472. The calculation of the index of the _path state in line 474 will always result in 1 due to the if condition, which will only allow the value of the _path.length state to be 2 at this point. The _path.length - 1 expression can be considered an invariant value and can be replaced with a constant to save the gas cost from the calculation.

OrderBook02.sol

```
462 function validateSwapOrderPriceWithTriggerAboveThreshold(
463   address[] memory _path,
464   uint256 _triggerRatio
465 ) public view returns (bool) {
466   if (_path.length != 2 && _path.length != 3) revert InvalidPathLength();
467 
468   // limit orders don't need this validation because minOut is enough
469   // so this validation handles scenarios for stop orders only
470   // when a user wants to swap when a price of tokenB increases relative to
tokenA
471   uint256 currentRatio;
472   if (_path.length == 2) {
473     address tokenA = _path[0];
474     address tokenB = _path[_path.length - 1];
```

```
    uint256 tokenAPrice;
    uint256 tokenBPrice;

    tokenAPrice = poolOracle.getMinPrice(tokenA);
    tokenBPrice = poolOracle.getMaxPrice(tokenB);

    currentRatio = (tokenBPrice * PRICE_PRECISION) / tokenAPrice;
  } else {
    address tokenA = _path[0];
    address tokenB = _path[1];
    address tokenC = _path[2];
    uint256 tokenAPrice;
    uint256 tokenBMinPrice;
    uint256 tokenBMaxPrice;
    uint256 tokenCPrice;

    tokenAPrice = poolOracle.getMinPrice(tokenA);
    tokenBMinPrice = poolOracle.getMinPrice(tokenB);
    tokenBMaxPrice = poolOracle.getMaxPrice(tokenB);
    tokenCPrice = poolOracle.getMaxPrice(tokenC);

    currentRatio =
      (tokenCPrice * tokenBMaxPrice * PRICE_PRECISION) /
      (tokenAPrice * tokenBMinPrice);
  }
  bool isValid = currentRatio > _triggerRatio;
  return isValid;
}
```

5.15.2. Remediation

Inspex suggests replacing the expression with a 1 literal.

OrderBook02.sol

```
function validateSwapOrderPriceWithTriggerAboveThreshold(
  address[] memory _path,
  uint256 _triggerRatio
) public view returns (bool) {
  if (_path.length != 2 && _path.length != 3) revert InvalidPathLength();

  // limit orders don't need this validation because minOut is enough
  // so this validation handles scenarios for stop orders only
  // when a user wants to swap when a price of tokenB increases relative to tokenA
  uint256 currentRatio;
  if (_path.length == 2) {
    address tokenA = _path[0];
    address tokenB = _path[1];
    uint256 tokenAPrice;
    uint256 tokenBPrice;

    tokenAPrice = poolOracle.getMinPrice(tokenA);
    tokenBPrice = poolOracle.getMaxPrice(tokenB);

    currentRatio = (tokenBPrice * PRICE_PRECISION) / tokenAPrice;
  } else {
    address tokenA = _path[0];
    address tokenB = _path[1];
    address tokenC = _path[2];
    uint256 tokenAPrice;
    uint256 tokenBMinPrice;
    uint256 tokenBMaxPrice;
    uint256 tokenCPrice;

    tokenAPrice = poolOracle.getMinPrice(tokenA);
    tokenBMinPrice = poolOracle.getMinPrice(tokenB);
    tokenBMaxPrice = poolOracle.getMaxPrice(tokenB);
    tokenCPrice = poolOracle.getMaxPrice(tokenC);

    currentRatio =
      (tokenCPrice * tokenBMaxPrice * PRICE_PRECISION) /
      (tokenAPrice * tokenBMinPrice);
  }
  bool isValid = currentRatio > _triggerRatio;
  return isValid;
}
```

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

inspex

CYBERSECURITY PROFESSIONAL SERVICE