# MDEX Integration

## Smart Contract Audit Report
## Prepared for Alpaca Finance

| | |
|---|---|
| **Date Issued:** | Sep 24, 2021 |
| **Project ID:** | AUDIT2021025 |
| **Version:** | v1.0 |
| **Confidentiality Level:** | Public |

## Report Information

| | |
|---|---|
| **Project ID** | AUDIT2021025 |
| **Version** | v1.0 |
| **Client** | Alpaca Finance |
| **Project** | MDEX Integration |
| **Auditor(s)** | Weerawat Pawanawiwat<br>Patipon Suwanbol |
| **Author** | Weerawat Pawanawiwat |
| **Reviewer** | Suvicha Buakhom |
| **Confidentiality Level** | Public |

## Version History

| Version | Date | Description | Author(s) |
|---|---|---|---|
| 1.0 | Sep 24, 2021 | Full report | Weerawat Pawanawiwat |

## Contact Information

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

# Table of Contents

# 1. Executive Summary

As requested by Alpaca Finance, Inspex team conducted an audit to verify the security posture of the MDEX Integration smart contracts between Sep 13, 2021 and Sep 17, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of MDEX Integration smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

## 1.1. Audit Result

In the initial audit, Inspex found 1 high, 2 medium, 4 low, 1 very low, and 1 info-severity issues. With the project team's prompt response, 1 high, 2 medium, 1 low, 1 very low, and 1 info-severity issues were resolved in the reassessment, while 3 low-severity issues were acknowledged by the team. Therefore, Inspex trusts that MDEX Integration smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

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

Approved by Inspex on Sep 24, 2021

## 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.

MDEX integration is composed of an optimized worker and multiple strategies that Alpaca Finance invented to help the users to earn optimally. This integration brings the additional farming options (MDEX's pools) to the users. With MDEX's pools supported on Alpaca Finance, the users can now perform leverage yield farming on MDEX's pools through Alpaca Finance to maximize the rewards, boosting the farming experience tremendously.

**Scope Information:**

| | |
|---|---|
| **Project Name** | MDEX Integration |
| **Website** | https://app.alpacafinance.org/farm |
| **Smart Contract Type** | Ethereum Smart Contract |
| **Chain** | Binance Smart Chain |
| **Programming Language** | Solidity |

**Audit Information:**

| | |
|---|---|
| **Audit Method** | Whitebox |
| **Audit Date** | Sep 13, 2021 - Sep 17, 2021 |
| **Reassessment Date** | Sep 24, 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: 6c973dd091d1d0da87c555e22a6cd865153b72f4)**

| Contract | Location (URL) |
|---|---|
| MdexWorker02 | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/6c973dd091/contracts/6/protocol/workers/mdex/MdexWorker02.sol |
| MdexRestrcitedStrategyAddBaseTokenOnly | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/6c973dd091/contracts/6/protocol/strategies/mdex/MdexRestrictedStrategyAddBaseTokenOnly.sol.sol |
| MdexRestrictedStrategyWithdrawMinimizeTrading | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/6c973dd091/contracts/6/protocol/strategies/mdex/MdexRestrictedStrategyWithdrawMinimizeTrading.sol |
| MdexRestrictedStrategyAddTwosideOptimal | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/6c973dd091/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyAddTwosidesOptimal.sol |
| MdexRestrictedStrategyLiquidate | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/6c973dd091/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyLiquidate.sol |
| MdexRestrictedStrategyPartialCloseLiquidate | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/6c973dd091/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyPartialCloseLiquidate.sol |
| MdexRestrictedStrategyPartialCloseMinimizeTrading | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/6c973dd091/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyPartialCloseMinimizeTrading.sol |

**Reassessment: (Commit: a3a14d27a4803afebdcf783c9ce8764b65f5587d)**

| Contract | Location (URL) |
|---|---|
| MdexWorker02 | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a3a14d27a4/contracts/6/protocol/workers/mdex/MdexWorker02.sol |
| MdexRestrcitedStrategyAddBaseTokenOnly | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a3a14d27a4/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyAddBaseTokenOnly.sol |
| MdexRestrictedStrategyWithdrawMinimizeTrading | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a3a14d27a4/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyWithdrawMinimizeTrading.sol |

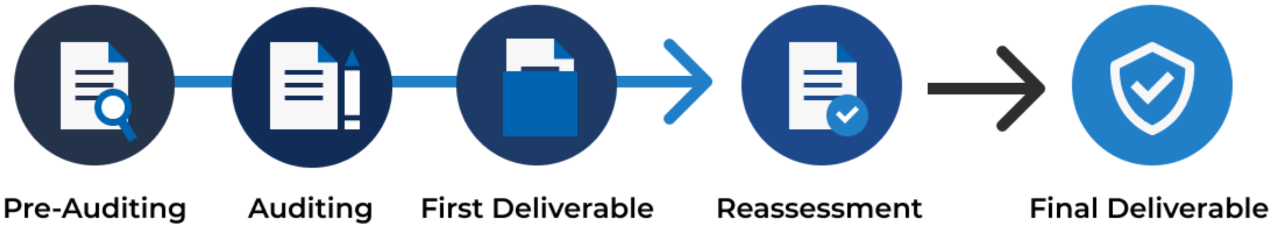
| | |
|---|---|
| MdexRestrictedStrategyAddTwosideOptimal | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a3a14d27a4/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyAddTwosidesOptimal.sol |
| MdexRestrictedStrategyLiquidate | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a3a14d27a4/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyLiquidate.sol |
| MdexRestrictedStrategyPartialCloseLiquidate | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a3a14d27a4/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyPartialCloseLiquidate.sol |
| MdexRestrictedStrategyPartialCloseMinimizeTrading | https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a3a14d27a4/contracts/6/protocol/strategies/mdex-restricted/MdexRestrictedStrategyPartialCloseMinimizeTrading.sol |

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

# 3. Methodology

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

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



## 3.1. Test Categories

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

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

## 3.2. Audit Items

The following audit items were checked during the auditing activity.

| General |
|---|
| Reentrancy Attack |
| Integer Overflows and Underflows |
| Unchecked Return Values for Low-Level Calls |
| Bad Randomness |
| Transaction Ordering Dependence |
| Time Manipulation |
| Short Address Attack |
| Outdated Compiler Version |
| Use of Known Vulnerable Component |
| Deprecated Solidity Features |
| Use of Deprecated Component |
| Loop with High Gas Consumption |
| Unauthorized Self-destruct |
| Redundant Fallback Function |
| **Advanced** |
| Business Logic Flaw |
| Ownership Takeover |
| Broken Access Control |
| Broken Authentication |
| Use of Upgradable Contract Design |
| Insufficient Logging for Privileged Functions |
| 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

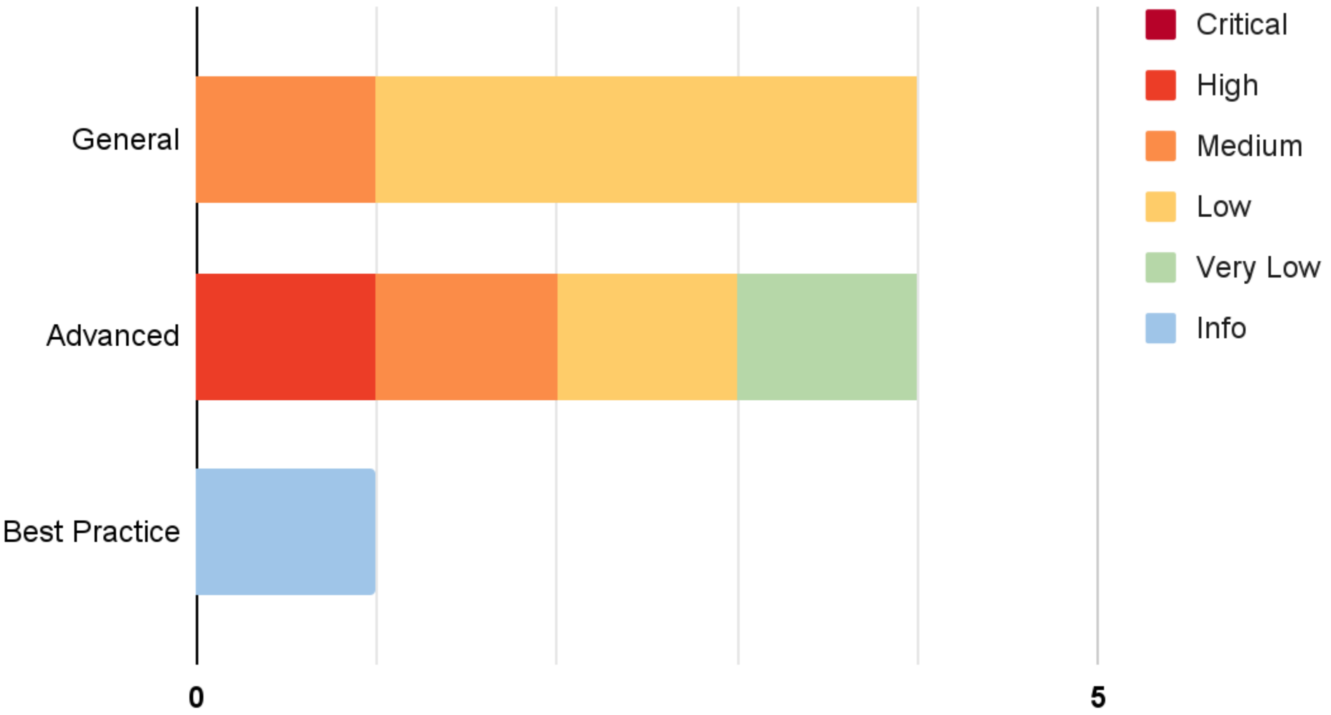
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

**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<br>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 9 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 | Advanced | High | Resolved * |
| IDX-002 | Improper Condition Checking in setRewardPath() Function | Advanced | Medium | Resolved |
| IDX-003 | Centralized Control of State Variable | General | Medium | Resolved * |
| IDX-004 | Transaction Ordering Dependence for Beneficial Vault Bounty | General | Low | Acknowledged |
| IDX-005 | Transaction Ordering Dependence for Reinvestment Balance | General | Low | Acknowledged |
| IDX-006 | Improper Transfer of Reinvest Token | Advanced | Low | Resolved |
| IDX-007 | Outdated Compiler Version | General | Low | Acknowledged |
| IDX-008 | Insufficient Logging for Privileged Functions | Advanced | Very Low | Resolved |
| IDX-009 | Improper Function Visibility | 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** | MdexRestrictedStrategyAddBaseTokenOnly<br>MdexRestrictedStrategyWithdrawMinimizeTrading<br>MdexRestrictedStrategyAddTwosidesOptimal<br>MdexRestrictedStrategyLiquidate<br>MdexRestrictedStrategyPartialCloseLiquidate<br>MdexRestrictedStrategyPartialCloseMinimizeTrading<br>MdexWorker02 |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-284: Improper Access Control |
| **Risk** | **Severity:** High<br>**Impact:** High<br>The logic of affected contracts can be arbitrarily changed. This allows the proxy contract owner to perform malicious actions e.g., stealing the users' funds.<br>**Likelihood:** Medium<br>Only the proxy contract owner can perform this action, but there is no restriction to prevent the owner from performing this attack, and it is profitable for the contract owner. |
| **Status** | Resolved *<br>Alpaca Finance team has confirmed that the upgradable contracts will be owned by the `ProxyAdmin` contract which is owned by the `Timelock` contract. This means any action that would occur to the upgradeable contracts will be able to be monitored by the community conveniently.<br>Since the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are owned by the `ProxyAdmin` contract which is under 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 upgradable.

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 be notified of potential changes being done on the smart contracts.

## 5.2 Improper Condition Checking in setRewardPath() Function

| | |
|---|---|
| **ID** | IDX-002 |
| **Target** | MdexWorker02 |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-755: Improper Handling of Exceptional Conditions |
| **Risk** | **Severity:** Medium<br>**Impact:** High<br>The `work()` function will be unusable since the `_reinvest()` function will revert the transaction due to the swapping of invalid tokens, resulting in failed executions on all strategies, including the users' close position transactions.<br>**Likelihood:** Low<br>Only the contract owner can set the reward path, and it is unlikely that the reward path will be set incorrectly because it is not profitable, resulting in a low motivation for the attack. |
| **Status** | Resolved<br>Alpaca Finance team has resolved this issue as suggested in commit `67181dd7e3b4c91fe7919ab626165fd329ff3969`. |

### 5.2.1 Description

In the `MdexWorker02` contract, the `_rewardToBeneficialVault()` function is called to send a bounty from reinvesting to the beneficial vault. Since the wanted token of the beneficial vault (`beneficialVaultToken`) is not the token that the `MdexWorker02` contract has, it is required to swap the reward token to the wanted token (`rewardPath`) before sending it.

**MdexWorker02.sol**

```
/// @dev Some portion of a bounty from reinvest will be sent to beneficialVault to increase the size of totalToken.
/// @param _beneficialVaultBounty - The amount of MDX to be swapped to BTOKEN & send back to the Vault.
/// @param _callerBalance - The balance that is owned by the msg.sender within the execution scope.
function _rewardToBeneficialVault(uint256 _beneficialVaultBounty, uint256 _callerBalance) internal {
    /// 1. read base token from beneficialVault
    address beneficialVaultToken = beneficialVault.token();
    /// 2. swap reward token to beneficialVaultToken
    uint256[] memory amounts =
        router.swapExactTokensForTokens(_beneficialVaultBounty, 0, rewardPath, address(this), now);
    /// 3. if beneficialvault token not equal to baseToken regardless of a caller balance, can directly transfer to beneficial vault
    /// otherwise, need to keep it as a buybackAmount,
    /// since beneficial vault is the same as the calling vault, it will think of this reward as a back amount to paydebt/ sending back to a position owner
    if (beneficialVaultToken != baseToken) {
        buybackAmount = 0;
        beneficialVaultToken.safeTransfer(address(beneficialVault), beneficialVaultToken.myBalance());
        emit BeneficialVaultTokenBuyback(msg.sender, beneficialVault, amounts[amounts.length - 1]);
    } else {
        buybackAmount = beneficialVaultToken.myBalance().sub(_callerBalance);
    }
}
```

The `rewardPath` can be set from the `setRewardPath()` function. However, the `rewardPath` can be set incorrectly because the checking mechanism validates the current value of `rewardPath` instead of the new one (`_rewardPath`).

**MdexWorker02.sol**

```
/// @dev Set a new reward path. In case that the liquidity of the reward path is changed.
/// @param _rewardPath The new reward path.
function setRewardPath(address[] calldata _rewardPath) external onlyOwner {
    require(rewardPath.length >= 2, "MdexWorker02::setRewardPath:: rewardPath length must be >= 2");
    require(
        rewardPath[0] == mdx && rewardPath[rewardPath.length - 1] == beneficialVault.token(),
        "MdexWorker02::setRewardPath:: rewardPath must start with MDX and end with beneficialVault token"
    );

    rewardPath = _rewardPath;

    emit SetRewardPath(msg.sender, _rewardPath);
}
```

This means improperly set value can potentially cause the contract to be unusable, for example, setting the `rewardPath[0]` as any token other than `mdx` will cause transactions that execute `_rewardToBeneficialVault()` to be reverted since there is no other token in the contract to be swapped. This results in the executions of all strategies through `work()` function to fail, including the openings or closings of positions by the users.

### 5.2.2 Remediation

Inspex suggests modifying the checking mechanism to validate the value of the `_rewardPath` parameter instead.

**MdexWorker02.sol**

```
/// @dev Set a new reward path. In case that the liquidity of the reward path is changed.
/// @param _rewardPath The new reward path.
function setRewardPath(address[] calldata _rewardPath) external onlyOwner {
    require(_rewardPath.length >= 2, "MdexWorker02::setRewardPath:: _rewardPath length must be >= 2");
    require(
        _rewardPath[0] == mdx && _rewardPath[_rewardPath.length - 1] ==
beneficialVault.token(),
        "MdexWorker02::setRewardPath:: _rewardPath must start with MDX and end
with beneficialVault token"
    );

    rewardPath = _rewardPath;

    emit SetRewardPath(msg.sender, _rewardPath);
}
```

## 5.3 Centralized Control of State Variable

| | |
|---|---|
| **ID** | IDX-003 |
| **Target** | MdexRestrictedStrategyAddBaseTokenOnly<br>MdexRestrictedStrategyWithdrawMinimizeTrading<br>MdexRestrictedStrategyAddTwosidesOptimal<br>MdexRestrictedStrategyLiquidate<br>MdexRestrictedStrategyPartialCloseLiquidate<br>MdexRestrictedStrategyPartialCloseMinimizeTrading<br>MdexWorker02 |
| **Category** | General Smart Contract Vulnerability |
| **CWE** | CWE-710: Improper Adherence to Coding Standard |
| **Risk** | **Severity:** Medium<br>**Impact:** Medium<br>The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users.<br>**Likelihood:** Medium<br>There is nothing to restrict the changes from being done; however, these actions can only be performed by the contract owner. |
| **Status** | Resolved *<br>Alpaca Finance team has confirmed that the contracts will be under the `Timelock` contract as same as other contracts on Alpaca Finance.<br>Since 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. |

### 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 |
|---|---|---|---|
| MdexRestrictedStrategyAddBaseTokenOnly.sol (L:110) | MdexRestrictedStrategyAddBaseTokenOnly | setWorkersOk() | onlyOwner |

| | | | |
|---|---|---|---|
| MdexRestrictedStrategyAddBaseTokenOnly.sol (L:137) | MdexRestrictedStrategyAddBaseTokenOnly | withdrawTradingRewards() | onlyOwner |
| MdexRestrictedStrategyWithdrawMinimizeTrading.sol (L:128) | MdexRestrictedStrategyWithdrawMinimizeTrading | setWorkersOk() | onlyOwner |
| MdexRestrictedStrategyWithdrawMinimizeTrading.sol (L:136) | MdexRestrictedStrategyWithdrawMinimizeTrading | withdrawTradingRewards() | onlyOwner |
| MdexRestrictedStrategyAddTwosidesOptimal.sol (L:177) | MdexRestrictedStrategyAddTwosidesOptimal | setWorkersOk() | onlyOwner |
| MdexRestrictedStrategyAddTwosidesOptimal.sol (L:185) | MdexRestrictedStrategyAddTwosidesOptimal | withdrawTradingRewards() | onlyOwner |
| MdexRestrictedStrategyLiquidate.sol (L:93) | MdexRestrictedStrategyLiquidate | setWorkersOk() | onlyOwner |
| MdexRestrictedStrategyLiquidate.sol (L:99) | MdexRestrictedStrategyLiquidate | withdrawTradingRewards() | onlyOwner |
| MdexRestrictedStrategyPartialCloseLiquidate.sol (L:110) | MdexRestrictedStrategyPartialCloseLiquidate | setWorkersOk() | onlyOwner |
| MdexRestrictedStrategyPartialCloseLiquidate.sol (L:118) | MdexRestrictedStrategyPartialCloseLiquidate | withdrawTradingRewards() | onlyOwner |
| MdexRestrictedStrategyPartialCloseMinimizeTrading.sol (L:153) | MdexRestrictedStrategyPartialCloseMinimizeTrading | setWorkersOk() | onlyOwner |
| MdexRestrictedStrategyPartialCloseMinimizeTrading.sol (L:161) | MdexRestrictedStrategyPartialCloseMinimizeTrading | withdrawTradingRewards() | onlyOwner |
| MdexWorker02.sol (L:447) | MdexWorker02 | setReinvestConfig() | onlyOwner |
| MdexWorker02.sol (L:471) | MdexWorker02 | setMaxReinvestBountyBps() | onlyOwner |
| MdexWorker02.sol | MdexWorker02 | setStrategyOk() | onlyOwner |

| | | | |
|---|---|---|---|
| (L:489) | | | |
| MdexWorker02.sol (L:501) | MdexWorker02 | setReinvestorOk() | onlyOwner |
| MdexWorker02.sol (L:512) | MdexWorker02 | setRewardPath() | onlyOwner |
| MdexWorker02.sol (L:527) | MdexWorker02 | setCriticalStrategies() | onlyOwner |
| MdexWorker02.sol (L:537) | MdeaxWorker02 | setTreasuryConfig() | onlyOwner |
| MdexWorker02.sol (L:553) | MdexWorker02 | setBeneficialVaultConfig() | onlyOwner |
| MdexWorker02.sol (L:579) | MdexWorker02 | withdrawTradingRewards() | onlyOwner |
| @openzeppelin/contracts-ethereum-package/contracts/access/Ownable.sol (L:63) | MdexRestrictedStrategyAddBaseTokenOnly, MdexRestrictedStrategyWithdrawMinimizeTrading, MdexRestrictedStrategyAddTwosidesOptimal, MdexRestrictedStrategyLiquidate, MdexRestrictedStrategyPartialCloseLiquidate, MdexRestrictedStrategyPartialCloseMinimizeTrading, MdexWorker02 | renounceOwnership() | onlyOwner |
| @openzeppelin/contracts-ethereum-package/contracts/access/Ownable.sol (L:72) | MdexRestrictedStrategyAddBaseTokenOnly, MdexRestrictedStrategyWithdrawMinimizeTrading, MdexRestrictedStrategyAddTwosidesOptimal, MdexRestrictedStrategyLiquidate, MdexRestrictedStrategyPartialCloseLiquidate, MdexRestrictedStrategyPartialCloseMinimizeTrading, MdexWorker02 | transferOwnership() | onlyOwner |

Please note that the `OwnableUpgradeSafe` contract is inherited from OpenZeppelin's library by all affected contracts.

### 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 limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a Timelock contract to delay the changes for a sufficient amount of time, e.g., 24 hours

## 5.4 Transaction Ordering Dependence for Beneficial Vault Bounty

| | |
|---|---|
| **ID** | IDX-004 |
| **Target** | MdexWorker02 |
| **Category** | General Smart Contract Vulnerability |
| **CWE** | CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') |
| **Risk** | **Severity:** Low<br>**Impact:** Medium<br>The front-running attack can be performed, resulting in a bad swapping rate for the beneficial vault bounty.<br>**Likelihood:** Low<br>It is easy to perform the attack. However, with a low profit, there is low motivation to attack with this vulnerability. |
| **Status** | **Acknowledged**<br>Alpaca Finance team has acknowledged this issue since the reward amount to motivate the transaction ordering is extremely small, so it is not worth introducing more complexity to the codebase. |

### 5.4.1 Description

In the `MdexWorker02` contract, the `_reinvest()` function is used to claim and reinvest the farming reward. This function can be executed by anyone through the `work()` function, or executed by the reinvestor through the `reinvest()` function. A part of the reward is charged as a fee and swapped to another token in the `_rewardToBeneficialVault()` function at line 233.

**MdexWorker02.sol**

```
209 /// @dev internal method for reinvest.
210 /// @param _treasuryAccount - The account to receive reinvest fees.
211 /// @param _treasuryBountyBps - The fees in BPS that will be charged for reinvest.
212 /// @param _callerBalance - The balance that is owned by the msg.sender within the execution scope.
213 /// @param _reinvestThreshold - The threshold to be reinvested if reward pass over.
214 function _reinvest(
215     address _treasuryAccount,
216     uint256 _treasuryBountyBps,
217     uint256 _callerBalance,
218     uint256 _reinvestThreshold
```

```
219 ) internal {
220     // 1. Withdraw all the rewards. Return if reward <= _reinvestThreshold.
221     bscPool.withdraw(pid, 0);
222     uint256 reward = mdx.balanceOf(address(this));
223     if (reward <= _reinvestThreshold) return;
224 
225     // 2. Approve tokens
226     mdx.safeApprove(address(router), uint256(-1));
227     address(lpToken).safeApprove(address(bscPool), uint256(-1));
228 
229     // 3. Send the reward bounty to the _treasuryAccount.
230     uint256 bounty = reward.mul(_treasuryBountyBps) / 10000;
231     if (bounty > 0) {
232         uint256 beneficialVaultBounty = bounty.mul(beneficialVaultBountyBps) / 10000;
233         if (beneficialVaultBounty > 0) _rewardToBeneficialVault(beneficialVaultBounty, _callerBalance);
234         mdx.safeTransfer(_treasuryAccount, bounty.sub(beneficialVaultBounty));
235     }
```

In the `_rewardToBeneficialVault()` function, the reward token ($MDX) is swapped to another token at line 345 with the minimum return amount set as 0, and that token is then sent to the beneficial vault at line 351.

**MdexWorker02.sol**

```
337 /// @dev Some portion of a bounty from reinvest will be sent to beneficialVault to increase the size of totalToken.
338 /// @param _beneficialVaultBounty - The amount of MDX to be swapped to BTOKEN & send back to the Vault.
339 /// @param _callerBalance - The balance that is owned by the msg.sender within the execution scope.
340 function _rewardToBeneficialVault(uint256 _beneficialVaultBounty, uint256 _callerBalance) internal {
341     /// 1. read base token from beneficialVault
342     address beneficialVaultToken = beneficialVault.token();
343     /// 2. swap reward token to beneficialVaultToken
344     uint256[] memory amounts =
345         router.swapExactTokensForTokens(_beneficialVaultBounty, 0, rewardPath, address(this), now);
346     /// 3. if beneficialvault token not equal to baseToken regardless of a caller balance, can directly transfer to beneficial vault
347     /// otherwise, need to keep it as a buybackAmount,
348     /// since beneficial vault is the same as the calling vault, it will think of this reward as a back amount to paydebt/ sending back to a position owner
349     if (beneficialVaultToken != baseToken) {
350         buybackAmount = 0;
```

```
        beneficialVaultToken.safeTransfer(address(beneficialVault), beneficialVaultToken.myBalance());
        emit BeneficialVaultTokenBuyback(msg.sender, beneficialVault, amounts[amounts.length - 1]);
    } else {
        buybackAmount = beneficialVaultToken.myBalance().sub(_callerBalance);
    }
}
```

This allows a front-running attack to be done, causing the output amount to be less than what it should be, causing the beneficial vault to gain less reward.

### 5.4.2 Remediation

Inspex suggests implementing a price oracle and using the price from the oracle to calculate the acceptable slippage. As an example, TWAP oracle can be used to get the price of the token pair from the on-chain data (https://docs.uniswap.org/protocol/V2/concepts/core-concepts/oracles).

## 5.5 Transaction Ordering Dependence for Reinvestment Balance

| | |
|---|---|
| **ID** | IDX-005 |
| **Target** | MdexWorker02 |
| **Category** | General Smart Contract Vulnerability |
| **CWE** | CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') |
| **Risk** | **Severity:** Low<br>**Impact:** Medium<br>The front-running attack can be performed, resulting in a bad swapping rate for the reinvestment and lower reward for the platform users.<br>**Likelihood:** Low<br>It is easy to perform the attack. However, with a low profit, there is low motivation to attack with this vulnerability. |
| **Status** | **Acknowledged**<br>Alpaca Finance team has acknowledged this issue since the reward amount to motivate the transaction ordering is extremely small, so it is not worth introducing more complexity to the codebase. |

### 5.5.1 Description

In the `MdexWorker02` contract, the `_reinvest()` function is used to claim and reinvest the farming reward. This function can be executed by anyone through the `work()` function, or executed by the reinvestor through the `reinvest()` function. A part of the reward is charged as a fee and sent to the treasury account or the reinvestor. After sending the fee, the rest of the reward is swapped to the base token in line 238 with the minimum swapping output amount set to 0, and sent to the strategy to get the LP token at line 241 and 242.

**MdexWorker02.sol**

```
237 // 4. Convert all the remaining rewards to BaseToken according to config path.
238 router.swapExactTokensForTokens(reward.sub(bounty), 0, getReinvestPath(),
    address(this), now);
239 
240 // 5. Use add Token strategy to convert all BaseToken without both caller
    balance and buyback amount to LP tokens.
241 baseToken.safeTransfer(address(addStrat),
    actualBaseTokenBalance().sub(_callerBalance));
242 addStrat.execute(address(0), 0, abi.encode(0));
243 
244 // 6. Stake LPs for more rewards
245 bscPool.deposit(pid, lpToken.balanceOf(address(this)));
```

The last parameter passed to the `addStrat.execute()` function is also set to 0. That parameter is used to check the minimum LP amount to gain from the liquidity provision at line 101, preventing excessive slippage from the token swapping.

**MdexRestrictedStrategyAddBaseTokenOnly.sol**

```
58 /// @dev Execute worker strategy. Take BaseToken. Return LP tokens.
59 /// @param data Extra calldata information passed along to this strategy.
60 function execute(
61     address, /* user */
62     uint256, /* debt */
63     bytes calldata data
64 ) external override onlyWhitelistedWorkers nonReentrant {
65     // 1. Find out what farming token we are dealing with and min additional LP tokens.
66     uint256 minLPAmount = abi.decode(data, (uint256));
67     IWorker worker = IWorker(msg.sender);
68     address baseToken = worker.baseToken();
69     address farmingToken = worker.farmingToken();
70     IPancakePair lpToken = IPancakePair(factory.getPair(farmingToken, baseToken));
71     // 2. Get trading fee of the pair from Mdex
72     uint256 fee = factory.getPairFees(address(lpToken));
73     // 3. Approve router to do their stuffs
74     baseToken.safeApprove(address(router), uint256(-1));
75     farmingToken.safeApprove(address(router), uint256(-1));
76     // 4. Compute the optimal amount of baseToken to be converted to farmingToken.
77     uint256 balance = baseToken.myBalance();
78     (uint256 r0, uint256 r1, ) = lpToken.getReserves();
79     uint256 rIn = lpToken.token0() == baseToken ? r0 : r1;
80     // find how many baseToken need to be converted to farmingToken
81     uint256 aIn = _calculateAIn(fee, rIn, balance);
82
83     // 5. Convert that portion of baseToken to farmingToken.
84     address[] memory path = new address[](2);
85     path[0] = baseToken;
86     path[1] = farmingToken;
87     router.swapExactTokensForTokens(aIn, 0, path, address(this), now);
88     // 6. Mint more LP tokens and return all LP tokens to the sender.
89     (, , uint256 moreLPAmount) =
90         router.addLiquidity(
91             baseToken,
92             farmingToken,
93             baseToken.myBalance(),
94             farmingToken.myBalance(),
95             0,
96             0,
            address(this),
            now
        );
    require(
        moreLPAmount >= minLPAmount,
        "MdexRestrictedStrategyAddBaseTokenOnly::execute:: insufficient LP tokens received"
    );
    address(lpToken).safeTransfer(msg.sender, lpToken.balanceOf(address(this)));
    // 7. Reset approval for safety reason
    baseToken.safeApprove(address(router), 0);
    farmingToken.safeApprove(address(router), 0);
}
```

Therefore, any amount of LP token is accepted, allowing front-running attack to be performed, resulting in less LP token for the reinvestment and reduced reward for the platform users.

### 5.5.2 Remediation

Inspex suggests implementing a price oracle and using the price from the oracle to calculate the acceptable slippage. As an example, TWAP oracle can be used to get the price of the token pair from the on-chain data (https://docs.uniswap.org/protocol/V2/concepts/core-concepts/oracles).

## 5.6 Improper Transfer of Reinvest Token

| | |
|---|---|
| **ID** | IDX-006 |
| **Target** | MdexWorker02 |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-840: Business Logic Errors |
| **Risk** | **Severity:** Low<br>**Impact:** Low<br>A small part of the $MDX token used for the reinvestment can be transferred out by the contract owner, resulting in less reward for the platform users.<br>**Likelihood:** Medium<br>Only the contract owner can perform this action; however, there is no restriction to prevent the owner from performing this attack, and it is profitable for the contract owner. |
| **Status** | Resolved<br>Alpaca Finance team has resolved this issue as suggested in commit `67181dd7e3b4c91fe7919ab626165fd329ff3969`. |

### 5.6.1 Description

In the `MdexWorker02` contract, the farming reward can be claimed and compounded using the `_reinvest()` function. This function can be executed by anyone through the `work()` function, or executed by the reinvestor through the `reinvest()` function. The reward is claimed in line 221, and the amount of reward token in the contract is checked whether it exceeds the minimum `_reinvestThreshold` or not in line 223. If the amount is not high enough, the reinvestment will be skipped, causing a small amount of reward token to stay in the contract.

**MdexWorker02.sol**

```
209 /// @dev internal method for reinvest.
210 /// @param _treasuryAccount - The account to receive reinvest fees.
211 /// @param _treasuryBountyBps - The fees in BPS that will be charged for reinvest.
212 /// @param _callerBalance - The balance that is owned by the msg.sender within the execution scope.
213 /// @param _reinvestThreshold - The threshold to be reinvested if reward pass over.
214 function _reinvest(
215     address _treasuryAccount,
216     uint256 _treasuryBountyBps,
217     uint256 _callerBalance,
218     uint256 _reinvestThreshold
```

```
219 ) internal {
220     // 1. Withdraw all the rewards. Return if reward <= _reinvestThreshold.
221     bscPool.withdraw(pid, 0);
222     uint256 reward = mdx.balanceOf(address(this));
223     if (reward <= _reinvestThreshold) return;
224 
225     // 2. Approve tokens
226     mdx.safeApprove(address(router), uint256(-1));
227     address(lpToken).safeApprove(address(bscPool), uint256(-1));
```

There is also the `withdrawTradingRewards()` function for the contract owner to call and claim the trading reward from the MDEX router. This function claims the reward at line 580, and transfer the whole balance of reward token ($MDX) in the contract to another address at line 581.

**MdexWorker02.sol**

```
577 /// @dev Withdraw trading all reward.
578 /// @param to The address to transfer trading reward to.
579 function withdrawTradingRewards(address to) external onlyOwner {
580     IMdexSwapMining(router.swapMining()).takerWithdraw();
581     mdx.safeTransfer(to, mdx.myBalance());
582 }
```

Since the reward claimed for the reinvestment can be left in the contract, transferring the whole balance of the reward token will include that amount, and cause the platform users to gain less reward due to the tokens transferred out by the contract owner.

### 5.6.2 Remediation

Inspex suggests checking the amount of the trading reward claimed in the `withdrawTradingRewards()` function, and transfer that amount instead of the whole balance, for example:

**MdexWorker02.sol**

```
577 /// @dev Withdraw trading all reward.
578 /// @param to The address to transfer trading reward to.
579 function withdrawTradingRewards(address to) external onlyOwner {
580     uint256 balanceBefore = mdx.myBalance();
581     IMdexSwapMining(router.swapMining()).takerWithdraw();
582     mdx.safeTransfer(to, mdx.myBalance().sub(balanceBefore));
583 }
```

## 5.7 Outdated Compiler Version

| | |
|---|---|
| ID | IDX-007 |
| Target | MdexRestrictedStrategyAddBaseTokenOnly<br>MdexRestrictedStrategyWithdrawMinimizeTrading<br>MdexRestrictedStrategyAddTwosidesOptimal<br>MdexRestrictedStrategyLiquidate<br>MdexRestrictedStrategyPartialCloseLiquidate<br>MdexRestrictedStrategyPartialCloseMinimizeTrading<br>MdexWorker02 |
| Category | General Smart Contract Vulnerability |
| CWE | CWE-1104: Use of Unmaintained Third Party Components |
| Risk | **Severity:** Low<br>**Impact:** Low<br>From the list of known Solidity bugs, direct impact cannot be caused from those bugs themselves.<br>**Likelihood:** Low<br>From the list of known Solidity bugs, it is very unlikely that those bugs would affect these smart contracts. |
| Status | **Acknowledged**<br>Alpaca Finance team has acknowledged this issue as all contracts and dependencies that Alpaca Finance team are using are based on Solidity version 0.6.6. |

### 5.7.1 Description

The Solidity compiler versions specified in the smart contracts were outdated. These versions have publicly known inherent bugs[2] that may potentially be used to cause damage to the smart contracts or the users of the smart contracts.

**MdexWorker02.sol**

```
pragma solidity 0.6.6;
```

The following table contains all targets which the compiler version is outdated.

| Target | Version |
|---|---|
| MdexRestrictedStrategyAddBaseTokenOnly | 0.6.6 |
| MdexRestrictedStrategyWithdrawMinimizeTrading | 0.6.6 |
| MdexRestrictedStrategyAddTwosidesOptimal | 0.6.6 |

| | |
|---|---|
| MdexRestrictedStrategyLiquidate | 0.6.6 |
| MdexRestrictedStrategyPartialCloseLiquidate | 0.6.6 |
| MdexRestrictedStrategyPartialCloseMinimizeTrading | 0.6.6 |
| MdexWorker02 | 0.6.6 |

### 5.7.2 Remediation

Inspex suggests upgrading the Solidity compiler of all affected contracts to the latest stable version[3]. During the audit activity, the latest stable versions of Solidity compiler in major 0.6 is v0.6.12.

## 5.8 Insufficient Logging for Privileged Functions

| | |
|---|---|
| **ID** | IDX-008 |
| **Target** | MdexRestrictedStrategyAddBaseTokenOnly<br>MdexRestrictedStrategyWithdrawMinimizeTrading<br>MdexRestrictedStrategyAddTwosidesOptimal<br>MdexRestrictedStrategyLiquidate<br>MdexRestrictedStrategyPartialCloseLiquidate<br>MdexRestrictedStrategyPartialCloseMinimizeTrading<br>MdexWorker02 |
| **Category** | Advanced Smart Contract Vulnerability |
| **CWE** | CWE-778: Insufficient Logging |
| **Risk** | **Severity:** Very Low<br>**Impact:** Low<br>Privileged functions' executions cannot be monitored easily by the users.<br>**Likelihood:** Low<br>It is not likely that the execution of the privileged functions will be a malicious action. |
| **Status** | Resolved<br>Alpaca Finance team has resolved this issue as suggested in commit `a3a14d27a4803afebdcf783c9ce8764b65f5587d`. |

### 5.8.1 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 to the platform.

For example, the owner can set the whitelisted contract address, allowing it to call the `execute()` function by using the `setWorkersOk()` function in the `MdexRestrictedStrategyAddBaseTokenOnly` contract, and no event is emitted.

The privileged functions without sufficient logging are as follows:

| File | Contract | Function |
|---|---|---|
| MdexRestrictedStrategyAddBaseTokenOnly.sol (L:110) | MdexRestrictedStrategyAddBaseTokenOnly | setWorkersOk() |
| MdexRestrictedStrategyAddBaseTokenOnly.sol (L:137) | MdexRestrictedStrategyAddBaseTokenOnly | withdrawTradingRewards() |

| | | |
|---|---|---|
| MdexRestrictedStrategyWithdrawMinimizeTrading.sol (L:128) | MdexRestrictedStrategyWithdrawMinimizeTrading | setWorkersOk() |
| MdexRestrictedStrategyWithdrawMinimizeTrading.sol (L:136) | MdexRestrictedStrategyWithdrawMinimizeTrading | withdrawTradingRewards() |
| MdexRestrictedStrategyAddTwosidesOptimal.sol (L:177) | MdexRestrictedStrategyAddTwosidesOptimal | setWorkersOk() |
| MdexRestrictedStrategyAddTwosidesOptimal.sol (L:185) | MdexRestrictedStrategyAddTwosidesOptimal | withdrawTradingRewards() |
| MdexRestrictedStrategyLiquidate.sol (L:93) | MdexRestrictedStrategyLiquidate | setWorkersOk() |
| MdexRestrictedStrategyLiquidate.sol (L:99) | MdexRestrictedStrategyLiquidate | withdrawTradingRewards() |
| MdexRestrictedStrategyPartialCloseLiquidate.sol (L:110) | MdexRestrictedStrategyPartialCloseLiquidate | setWorkersOk() |
| MdexRestrictedStrategyPartialCloseLiquidate.sol (L:118) | MdexRestrictedStrategyPartialCloseLiquidate | withdrawTradingRewards() |
| MdexRestrictedStrategyPartialCloseMinimizeTrading.sol (L:153) | MdexRestrictedStrategyPartialCloseMinimizeTrading | setWorkersOk() |
| MdexRestrictedStrategyPartialCloseMinimizeTrading.sol (L:161) | MdexRestrictedStrategyPartialCloseMinimizeTrading | withdrawTradingRewards() |
| MdexWorker02.sol (L:579) | MdexWorker02 | withdrawTradingRewards() |

### 5.8.2 Remediation

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

**MdexRestrictedStrategyAddBaseTokenOnly.sol**

```
event SetWorkersOk(address indexed workerAddress, bool isOk);
function setWorkersOk(address[] calldata workers, bool isOk) external onlyOwner
{
    for (uint256 idx = 0; idx < workers.length; idx++) {
      okWorkers[workers[idx]] = isOk;
      emit SetWorkersOk(workers[idx], isOk);
    }
}
```

## 5.9 Improper Function Visibility

| | |
|---|---|
| **ID** | IDX-009 |
| **Target** | MdexRestrictedStrategyPartialCloseLiquidate |
| **Category** | Smart Contract Best Practice |
| **CWE** | CWE-710: Improper Adherence to Coding Standards |
| **Risk** | **Severity:** Info<br>**Impact: None**<br>**Likelihood: None** |
| **Status** | Resolved<br>Alpaca Finance team has resolved this issue as suggested in commit `a3a14d27a4803afebdcf783c9ce8764b65f5587d`. |

### 5.9.1 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.

The `initialize()` function of the `MdexRestrictedStrategyPartialCloseLiquidate` contract is set to public and it is never called from any internal function.

**MdexRestrictedStrategyPartialCloseLiquidate.sol**

```
function initialize(IMdexRouter _router, address _mdx) public initializer {
    OwnableUpgradeSafe.__Ownable_init();
    ReentrancyGuardUpgradeSafe.__ReentrancyGuard_init();
    factory = IMdexFactory(_router.factory());
    router = _router;
    mdx = _mdx;
}
```

### 5.9.2 Remediation

Inspex suggests changing the `initialize()` function's visibility to `external` if they are not called from any internal function as shown below:

**MdexRestrictedStrategyPartialCloseLiquidate.sol**

```
function initialize(IMdexRouter _router, address _mdx) external initializer {
    OwnableUpgradeSafe.__Ownable_init();
    ReentrancyGuardUpgradeSafe.__ReentrancyGuard_init();
    factory = IMdexFactory(_router.factory());
    router = _router;
    mdx = _mdx;
}
```

# 6. Appendix

## 6.1. About Inspex

CYBERSECURITY PROFESSIONAL SERVICE

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

**Follow Us On:**

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

## 6.2. References

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

[2] “List of Known Bug — Solidity 0.6.6 documentation” [Online]. Available: https://docs.soliditylang.org/en/v0.6.6/bugs.html. [Accessed: 15-September-2021]

[3] “Releases — Ethereum Solidity Releases” [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 15-September-2021]

inspex

CYBERSECURITY PROFESSIONAL SERVICE