AutoCompoundVault

Smart Contract Audit Report
Prepared for MondayClub

Date Issued:	Mar 17, 2022
Project ID:	AUDIT2022014
Version:	v1.0
Confidentiality Level:	Public

Report Information

Project ID	AUDIT2022014
Version	v1.0
Client	MondayClub
Project	AutoCompoundVault
Auditor(s)	Natsasit Jirathammanuwat Wachirawit Kanpanluk
Author(s)	Natsasit Jirathammanuwat
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Mar 17, 2022	Full report	Natsasit Jirathammanuwat

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 MondayClub, Inspex team conducted an audit to verify the security posture of the AutoCompoundVault smart contracts on Mar 8, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of AutoCompoundVault 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, 1 medium, 1 low, 1 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 AutoCompoundVault 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 17, 2022

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

Monday Club is an ecosystem for the workforce, to make working life fun again. Monday Club provides a bridge between traditional working life and life on the blockchain for regular workers - through Avatar NFTs, MetaOffice, Crypto payrolls & simplified DeFi investments.

AutoCompoundVault is a contract that provides functionality to harvest and compound the reward automatically on the yield farming platform. The performance fee will be collected and distributed to harvesters.

Scope Information:

Project Name	AutoCompoundVault
Website	https://mondayclub.io/
Smart Contract Type	Ethereum Smart Contract
Chain	BNB Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Mar 8, 2022
Reassessment Date	Mar 16, 2022

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: 16a17e4d60d174333b0041ad30d7c803f170bd01)

Contract	Location (URL)
AutoCompoundVault	https://github.com/mondayclub/mondayclub/blob/16a17e4d60/contracts/vaults/AutoCompoundVault.sol
StrategyCommonChefLP	https://github.com/mondayclub/mondayclub/blob/16a17e4d60/contracts/strategies/Common/StrategyCommonChefLP.sol

Reassessment: (Commit: 9ec29e0bb41a6bbef5484976502c924f4557ca02)

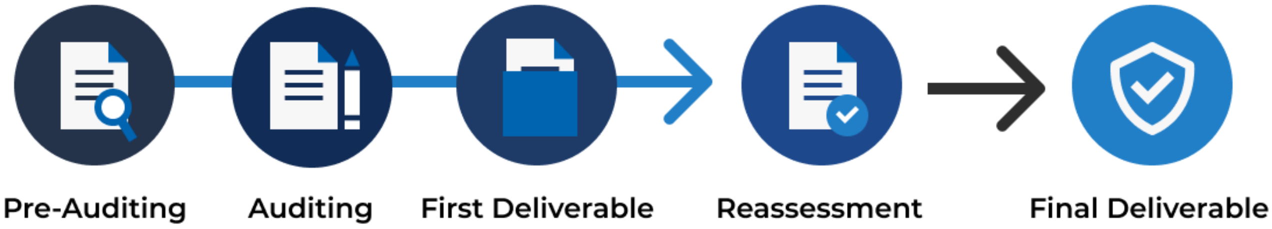
Contract	Location (URL)
AutoCompoundVault	https://github.com/mondayclub/mondayclub/blob/9ec29e0bb4/contracts/vaults/AutoCompoundVault.sol
StrategyCommonChefLP	https://github.com/mondayclub/mondayclub/blob/9ec29e0bb4/contracts/strategies/Common/StrategyCommonChefLP.sol

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

3. Methodology

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

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



3.1. Test Categories

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

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

3.2. Audit Items

The following audit items were checked during the auditing activity.

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

Improper Kill-Switch Mechanism
Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology (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

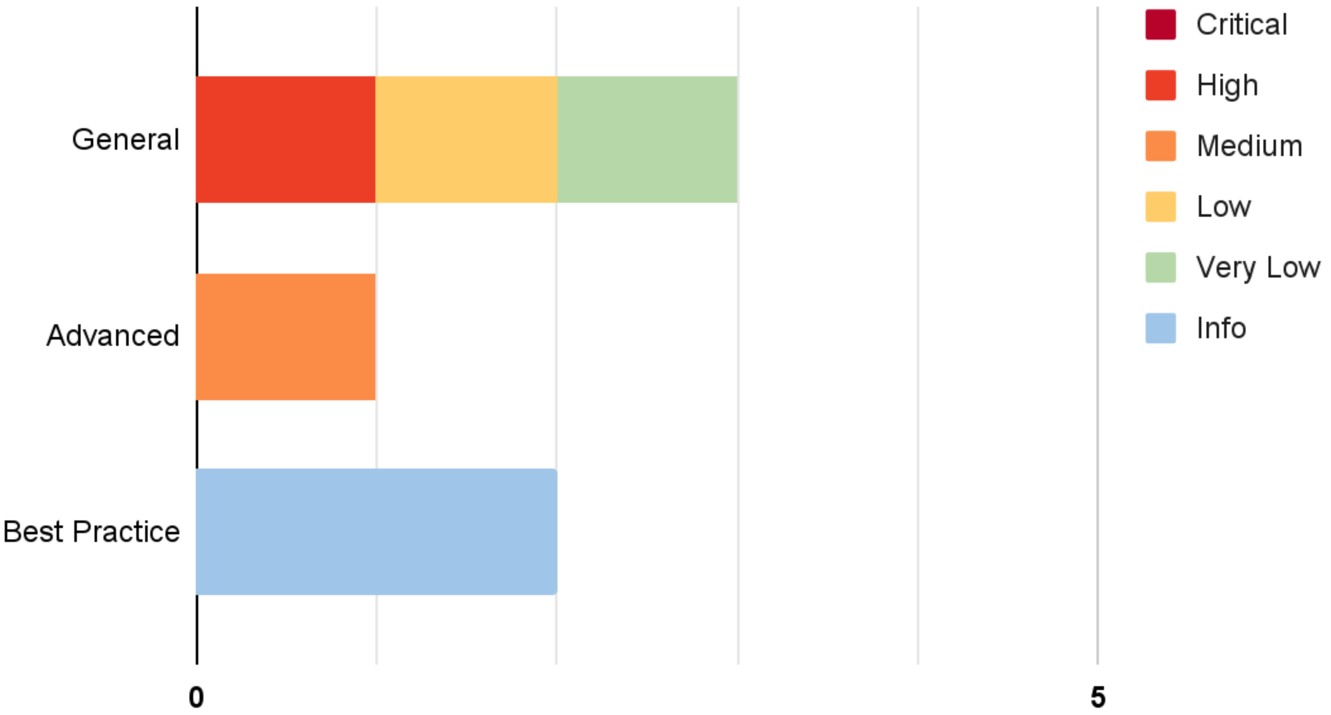
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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

From the assessments, Inspex has found 6 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	Centralized Control of State Variable	General	High	Resolved *
IDX-002	Improper Reward Distribution	Advanced	Medium	Resolved *
IDX-003	Transaction Ordering Dependence	General	Low	Resolved *
IDX-004	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-005	Inexplicit Solidity Compiler Version	Best Practice	Info	Resolved
IDX-006	Improper Function Visibility	Best Practice	Info	Resolved

* The mitigations or clarifications by MondayClub Team can be found in Chapter 5.

5. Detailed Findings Information

5.1. Centralized Control of State Variable

ID	IDX-001
Target	StrategyCommonChefLP
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 * MondayClub team has confirmed that the privilege function will be called through the `Timelock` contract. This means any action that would occur to the privilege function will be able to be monitored by the community conveniently. In the long term, MondayClub team plan to transfer ownership to the governance contract once it is implemented. However, as the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are under the effect of the `Timelock` contract before using them.

5.1.1. Description

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

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier
StrategyCommonChefLP.sol (L:228)	StrategyCommonChefLP	setHarvestOnDeposit()	onlyManager
StrategyCommonChefLP.sol (L:238)	StrategyCommonChefLP	setShouldGasThrottle()	onlyManager
FeeManager.sol(L:19)	StrategyCommonChefLP	setCallFee()	onlyManager

FeeManager.sol(L:26)	StrategyCommonChefLP	setWithdrawalFee()	onlyManager
StratManager.sol(L:49)	StrategyCommonChefLP	setKeeper()	onlyManager
StratManager.sol(L:57)	StrategyCommonChefLP	setUnirouter()	onlyOwner
StratManager.sol(L:65)	StrategyCommonChefLP	setVault()	onlyOwner
StratManager.sol(L:73)	StrategyCommonChefLP	setMondayFeeRecipient()	onlyOwner

5.1.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. Inspex suggests removing the state modification functions. 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 reasonable amount of time, e.g., 24 hours

Please keep in mind that if the timelock mechanism is chosen, the `pause()` and `unpause()` functions will be affected as well. Therefore, these function modifiers should be changed to another role modifier, so that the function callings will no longer be affected by the timelock.

5.2. Improper Reward Distribution

ID	IDX-002
Target	AutoCompoundVault StrategyCommonChefLP
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium A part of the compounded rewards could be claimed by a user without any prior token deposited, causing the other users to gain less compounded rewards. **Likelihood:** Medium It is likely that the `_harvest()` function will not be executed to harvest the pending farming reward before depositing the token.
Status	Resolved * MondayClub team has clarified that they have a withdrawal fee in place to prevent users from trying to take advantage. They will implement the auto-compound bot to compound rewards periodically, which makes the pending rewards always at a low value.

5.2.1. Description

In the `AutoCompoundVault` contract, the `deposit()` function is used for staking token into the contract.

AutoCompoundVault.sol

```
function deposit(uint _amount) public nonReentrant {
    strategy.beforeDeposit();

    uint256 _pool = balance();
    want().safeTransferFrom(msg.sender, address(this), _amount);
    earn();
    uint256 _after = balance();
    _amount = _after.sub(_pool); // Additional check for deflationary tokens
    uint256 shares = 0;
    if (totalSupply() == 0) {
        shares = _amount;
    } else {
        shares = (_amount.mul(totalSupply())).div(_pool);
    }
    _mint(msg.sender, shares);
}
```

As shown previously, the `strategy.beforeDeposit()` is called from the `deposit()` function in line 95, if the platform manager set the `harvestOnDeposit` state to `true`, the `_harvest` function is also called in order to update the total LP token (`balance()`) in line 116 as shown below.

StrategyCommonChefLP.sol

```
113  function beforeDeposit() external override {
114      if (harvestOnDeposit) {
115          require(msg.sender == vault, "!vault");
116          _harvest(tx.origin);
117      }
118  }
```

The farming reward will be converted into LP tokens and deposited to the staking pool as shown in lines 138 and 140.

StrategyCommonChefLP.sol

```
133  function _harvest(address callFeeRecipient) internal whenNotPaused {
134      IMasterChef(chef).deposit(poolId, 0);
135      uint256 outputBal = IERC20(output).balanceOf(address(this));
136      if (outputBal > 0) {
137          chargeFees(callFeeRecipient);
138          addLiquidity();
139          uint256 wantHarvested = balanceOfWant();
140          deposit();
141
142          lastHarvest = block.timestamp;
143          emit StratHarvest(msg.sender, wantHarvested, balanceOf());
144      }
145  }
```

However, if the `harvestOnDeposit` state is `false`, the total LP token will not be updated and the calculation of the withdrawable amount will be miscalculated as shown in line 134.

AutoCompoundVault.sol

```
133  function withdraw(uint256 _shares) public {
134      uint256 r = (balance().mul(_shares)).div(totalSupply());
135      _burn(msg.sender, _shares);
136
137      uint b = want().balanceOf(address(this));
138      if (b < r) {
139          uint _withdraw = r.sub(b);
140          strategy.withdraw(_withdraw);
141          uint _after = want().balanceOf(address(this));
142          uint _diff = _after.sub(b);
143          if (_diff < _withdraw) {
```

```
            r = b.add(_diff);
        }
    }

    want().safeTransfer(msg.sender, r);
}
```

The following example scenario shows that user A also get a part of the compounded rewards (5 LP tokens) from the pending reward that was just harvested after user A has deposited:

Event	Total shares	Total LP balance	Pending Reward
Initial state	100.00	100.00	10.00
User A deposit 100 LP token and get 100 shares	200.00	200.00	10.00
Harvest reward and add LP token	200.00	210.00	0.00
User A withdraw 100 shares and get 105 LP token	100.00	105.00	0.00

This will cause the other users to gain less compounded rewards.

5.2.2. Remediation

Inspex suggests always calling the `_harvest()` function before the **`deposit()`** function is called. This makes the **`deposit()`** function always harvest the pending reward and update the total of LP tokens before using it to calculate the user's withdrawable amount, for example:

StrategyCommonChefLP.sol

```
function beforeDeposit() external override {
    require(msg.sender == vault, "!vault");
    _harvest(tx.origin);
}
```

5.3. Transaction Ordering Dependence

ID	IDX-003
Target	StrategyCommonChefLP
Category	General Smart Contract Vulnerability
CWE	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization (‘Race Condition’)
Risk	**Severity:** Low **Impact:** Medium The front-running attack can be performed, resulting in a bad swapping rate for the reinvestment and lower reward for the platform users. **Likelihood:** Low It is easy to perform the attack. However, with a low profit, there is low motivation to attack with this vulnerability.
Status	Resolved * MondayClub team has confirmed that they will implement the auto-compound bot to compound rewards periodically, which makes the pending rewards always at a low value.

5.3.1. Description

In the `StrategyCommonChefLP` contract, the reward of the farming is compounded using the `_harvest()` function.

StrategyCommonChefLP.sol

```
function _harvest(address callFeeRecipient) internal whenNotPaused {
    IMasterChef(chef).deposit(poolId, 0);
    uint256 outputBal = IERC20(output).balanceOf(address(this));
    if (outputBal > 0) {
        chargeFees(callFeeRecipient);
        addLiquidity();
        uint256 wantHarvested = balanceOfWant();
        deposit();

        lastHarvest = block.timestamp;
        emit StratHarvest(msg.sender, wantHarvested, balanceOf());
    }
}
```

the addLiquidity() function performs token swapping using the IUniswapRouterETH(unirouter).swapExactTokensForTokens() function in line 166 and 170 to convert the farming reward to another token for the reinvestment.

StrategyCommonChefLP.sol

```
162 function addLiquidity() internal {
163     uint256 outputHalf = IERC20(output).balanceOf(address(this)).div(2);
164
165     if (lpToken0 != output) {
166         IUniswapRouterETH(unirouter).swapExactTokensForTokens(outputHalf, 0,
    outputToLp0Route, address(this), block.timestamp);
167     }
168
169     if (lpToken1 != output) {
            IUniswapRouterETH(unirouter).swapExactTokensForTokens(outputHalf, 0,
170 outputToLp1Route, address(this), block.timestamp);
171     }
172
173     uint256 lp0Bal = IERC20(lpToken0).balanceOf(address(this));
        uint256 lp1Bal = IERC20(lpToken1).balanceOf(address(this));
174     IUniswapRouterETH(unirouter).addLiquidity(lpToken0, lpToken1, lp0Bal,
175 lp1Bal, 1, 1, address(this), block.timestamp);
176 }
```

In the source code above, the swapping tolerance (amountOutMin) of the swapping function is set to 0. this allows the front-running attacker to be done, resulting in fewer tokens gained from the swapping.

5.3.2. Remediation

Inspex suggests calculating the expected swapping tolerance (amountOutMin) with the token price fetched from the price oracles and setting it to the amountOutMin parameter, for example:

StrategyCommonChefLP.sol

```
162 function addLiquidity() internal {
163     uint256 outputHalf = IERC20(output).balanceOf(address(this)).div(2);
164
165     if (lpToken0 != output) {
166         uint256 amountOutMin0 = calculateAmountOutMinFromOracle(outputHalf,
    outputToLp0Route);
            IUniswapRouterETH(unirouter).swapExactTokensForTokens(outputHalf,
167 amountOutMin0, outputToLp0Route, address(this), block.timestamp);
168     }
169
        if (lpToken1 != output) {
170         uint256 amountOutMin1 = calculateAmountOutMinFromOracle(outputHalf,
    outputToLp1Route);
```

```
        IUniswapRouterETH(unirouter).swapExactTokensForTokens(outputHalf, amountOutMin1, outputToLp1Route, address(this), block.timestamp);
    }

    uint256 lp0Bal = IERC20(lpToken0).balanceOf(address(this));
    uint256 lp1Bal = IERC20(lpToken1).balanceOf(address(this));
    IUniswapRouterETH(unirouter).addLiquidity(lpToken0, lpToken1, lp0Bal, lp1Bal, 1, 1, address(this), block.timestamp);
}
```

5.4. Insufficient Logging for Privileged Functions

ID	IDX-004
Target	AutoCompoundVault StrategyCommonChefLP
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 unlikely that the execution of the privileged functions will be a malicious action.
Status	Resolved MondayClub team has resolved this issue by adding events to the affected functions as suggested in commit `2f272fddb817206b1780f5ad8ef6b90e75e6bb56`.

5.4.1. 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 `onlyManager` privileged can set the `harvestOnDeposit` state to allow the users harvest the pending farming reward from the staking pool before deposit the token. This can be achieved by executing the `setHarvestOnDeposit()` function in the `StrategyCommonChefLP` contract, and no events are emitted.

StrategyCommonChefLP.sol

```
function setHarvestOnDeposit(bool _harvestOnDeposit) external onlyManager {
    harvestOnDeposit = _harvestOnDeposit;

    if (harvestOnDeposit) {
        setWithdrawalFee(0);
    } else {
        setWithdrawalFee(10);
    }
}
```

The privileged functions with insufficient logging are as follows:

File	Contract	Function	Modifier
AutoCompoundVault.sol (L:155)	AutoCompoundVault	inCaseTokensGetStuck()	onlyOwner
StrategyCommonChefLP.sol (L:194)	StrategyCommonChefLP	setPendingRewardsFunctionName()	onlyManager
StrategyCommonChefLP.sol (L:228)	StrategyCommonChefLP	setHarvestOnDeposit()	onlyManager
StrategyCommonChefLP.sol (L:238)	StrategyCommonChefLP	setShouldGasThrottle()	onlyManager
FeeManager.sol(L:19)	StrategyCommonChefLP	setCallFee()	onlyManager
FeeManager.sol(L:26)	StrategyCommonChefLP	setWithdrawalFee()	onlyManager
StratManager.sol(L:49)	StrategyCommonChefLP	setKeeper()	onlyManager
StratManager.sol(L:57)	StrategyCommonChefLP	setUnirouter()	onlyOwner
StratManager.sol(L:65)	StrategyCommonChefLP	setVault()	onlyOwner
StratManager.sol(L:73)	StrategyCommonChefLP	setMondayFeeRecipient()	onlyOwner

5.4.2. Remediation

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

StrategyCommonChefLP.sol

```
227 event SetHarvestOnDeposit(bool _harvestOnDeposit);
228 function setHarvestOnDeposit(bool _harvestOnDeposit) external onlyManager {
229     harvestOnDeposit = _harvestOnDeposit;
230
231     if (harvestOnDeposit) {
232         setWithdrawalFee(0);
233     } else {
234         setWithdrawalFee(10);
235     }
236     emit SetHarvestOnDeposit(_harvestOnDeposit);
237 }
```

5.5. Inexplicit Solidity Compiler Version

ID	IDX-005
Target	AutoCompoundVault StrategyCommonChefLP
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved MondayClub team has resolved this issue by updating the solidity compiler version of the affected contract as suggested in commit `9ec29e0bb41a6bbef5484976502c924f4557ca02`.

5.5.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues.

AutoCompoundVault.sol

```
pragma solidity ^0.8.4;
```

The following table contains all contracts that use inexplicit solidity compiler version.

Contract Name	Version
AutoCompoundVault	^0.8.4
StrategyCommonChefLP	^0.8.4

5.5.2. Remediation

Inspex suggests fixing the Solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is v0.8.12 (https://docs.soliditylang.org/en/v0.8.12/).

AutoCompoundVault.sol

```
pragma solidity 0.8.12;
```

5.6. Improper Function Visibility

ID	IDX-006
Target	StrategyCommonChefLP
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved MondayClub team has resolved this issue by changing the function visibility to the affected functions as suggested in commit `2f272fddb817206b1780f5ad8ef6b90e75e6bb56`.

5.6.1. 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 following source code shows that the `pause()` function in the `StrategyCommonChefLP` contract is set to public and it is never called from any internal function.

StrategyCommonChefLP.sol

```
242 function pause() public onlyManager {
243     _pause();
244
245     _removeAllowances();
246 }
```

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

File	Contract	Function
StrategyCommonChefLP.sol (L:242)	StrategyCommonChefLP	pause()
FeeManager.sol (L:19)	StrategyCommonChefLP	setCallFee()

5.6.2. Remediation

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

StrategyCommonChefLP.sol

```
function pause() external onlyManager {
    _pause();

    _removeAllowances();
}
```

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