Security Assessment

Alpaca Finance

May 13th, 2021

Summary

This report has been prepared for Alpaca Finance smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Alpaca Finance
Platform	BSC
Language	Solidity
Codebase	https://github.com/alpaca-finance/bsc-alpaca-contract
Commits	7b8389ac08f2025af8bad23af0ba7ea91ca94c26

Audit Summary

Delivery Date	May 13, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

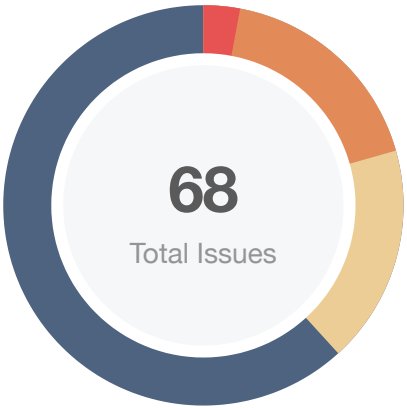
Total Issues	68
Critical	2
Major	12
Medium	0
Minor	12
Informational	42
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
TCK	Timelock.sol	16de91fefbf0e27dabd29e42d0acbb48ac984c75af7eae435a861ec1a03dc2e4
CIV	protocol/ConfigurableInterestVaultConfig.sol	493add62c1d0ee7d8c1d68ddf7faeb5deffc53e9a4235922d101b7f88705fd8b
DTC	protocol/DebtToken.sol	a33bcac6b84794199eda950ad3250c23545603086056b4b18e2c3ea875708b60
ITR	protocol/IbTokenRouter.sol	d33a9c3ac3963b667d6a075e289261bebf64585e96790cd9cc0e05a5f1b5e7e2
POC	protocol/PriceOracle.sol	8dbcf1a9d6d211c90321748f68a2c0f12ff7a7c1c2d893d8916a0f11c1af667d
SPO	protocol/SimplePriceOracle.sol	a3e0dcd9658393a438a0b7c2695b5fa3377960f1483d6ff45d33c08361c0d19f
SVC	protocol/SimpleVaultConfig.sol	55a019398538220cb322f42a8c586fa0d3610c8887848582538779b3c6c2f291
VCK	protocol/Vault.sol	9f4e4d7dccc96cb7ed332c735ac48d5542ee4331071367573eb94a643ddbad32
WNR	protocol/WNativeRelayer.sol	5051cd722f7cc808a1e07d4a74f1fdcb2172d159a61efe71af2652bda93de722
TSM	protocol/interest-models/TripleSlopeModel.sol	551f185011d9ea8e92e569d69d7b4565d1411d02bf250972c6797c52a5ebd293
SAB	protocol/strategies/pancakeswap/StrategyAddBaseTokenOnly.sol	23eb36adb140fa416995cab342f892242e79cbe9718384ef8e4e77a876e3caf7
SAT	protocol/strategies/pancakeswap/StrategyAddTwoSidesOptimal.sol	dc7834857ff509b541ded749ccc758b9ab6ff6e0d5a6f5a7ae14e64d78fe1c93
SLC	protocol/strategies/pancakeswap/StrategyLiquidate.sol	d7fd55da7caf84aecfbd28e5e91f342b43e51061386dae732b134aa034aea276
SWM	protocol/strategies/pancakeswap/StrategyWithdrawMinimizeTrading.sol	963506f7c30e026b6eeb3615e0cafb4ccff89116e0f710ac73496eda60c7c78e
PWC	protocol/workers/PancakeswapWorker.sol	af225614a9d85ba5e44335cff34b23c5d7b4ceca292aa7f4dc0b2f484b34be83
WCC	protocol/workers/WorkerConfig.sol	6493c40deac8b60134a45e259e3ad414052befb88d0ba097dce49054fb975005
ATC	token/AlpacaToken.sol	d0bafecf9d404ff13cd891a4e7c4b5b4e455bc9242e82dbfaf0b2beb15b99760
FLC	token/FairLaunch.sol	ebba35e56115b4fd7826828544f8d446cd93c2fa906c64eec8d87d1e9fb760b6
FLV	token/FairLaunchV2.sol	0d3220a3d061a99026d7427971c29d62a9a8797dd64b5f952d16533a73f5347e

ID	file	SHA256 Checksum
SCK	token/Shield.sol	c3c5e375e96b5238d728f86fb450cf4da4ac0df7220ab1e0e5a9addc7192013a
SAC	token/StronkAlpaca.sol	eb4c9cd117e24a3a1f77969a5fbefc665f5608e085acbca444fffe54d90b3d13
SAR	token/StronkAlpacaRelayer.sol	e35d7508eb42dffd005a9958dd6fe67ea756872671ff6dc0e9b3e236364e2231
LRC	token/lockers/LinearRelease.sol	8b16338d60ed0354682bbf7f702ad68557c2b1855577e2a63d564885505051b1
AMC	utils/AlpacaMath.sol	586e3f78a9a9b4706c1ab68e30b16ecb6a8aef7886fff95febbc79607c60d006
STC	utils/SafeToken.sol	6505be69ef107c99e07c3cdbcc1834517878f8dad34eb966ea027543609ed97a

Findings



Critical	**2** (2.94%)
Major	**12** (17.65%)
Medium	**0** (0.00%)
Minor	**12** (17.65%)
Informational	**42** (61.76%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
ATC-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
ATC-02	Unexpected Token Locking	Logical Issue	Informational	Resolved
ATC-03	Lack of State Update in `manualMint`	Logical Issue	Critical	Resolved
ATC-04	Centralization Risks I	**Centralization / Privilege**	**Major**	**Partially Resolved**
ATC-05	Centralization Risks II	**Centralization / Privilege**	**Major**	**Resolved**
ATC-06	Unused Function `setReleaseBlock`	Gas Optimization	Informational	Resolved
ATC-07	Variables Should Be Declared Constant	Gas Optimization	Informational	Resolved
CIV-01	Centralization Risks	**Centralization / Privilege**	**Major**	**Partially Resolved**
CIV-02	Function Should Be Declared External	Gas Optimization	Informational	Resolved
DTC-01	Lack of Allowance Check In `transferFrom`	Logical Issue	Critical	Resolved
DTC-02	Dead Code	Gas Optimization	Informational	Acknowledged
DTC-03	Function Should Be Declared External	Gas Optimization	Informational	Resolved
FLC-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
FLC-02	Lack of Return Value Handling	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
FLC-03	Lack of Checks for Reentrancy	Logical Issue	Major	Partially Resolved
FLC-04	Division Before Multiplication	Mathematical Operations	Informational	Partially Resolved
FLV-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
FLV-02	Lack of Return Value Handling	Logical Issue	Minor	Resolved
FLV-03	Lack of Checks for Reentrancy	Logical Issue	Major	Partially Resolved
FLV-04	Division Before Multiplication	Mathematical Operations	Informational	Partially Resolved
ITR-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
LRC-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
LRC-02	Redundant Data Structure	Gas Optimization	Informational	Resolved
PWC-01	Lack of Return Value Handling	Logical Issue	Minor	Partially Resolved
PWC-02	Lack of Event for Significant Transaction	Data Flow	Informational	Resolved
PWC-03	Function Should Be Declared External	Gas Optimization	Informational	Resolved
PWC-04	Centralization Risks	**Centralization / Privilege**	**Major**	**Partially Resolved**
PWC-05	Non-Optimal Parameters Passed to Strategy	Logical Issue	Minor	Partially Resolved
SAB-01	Unused Variable	Dead Code	Informational	Resolved
SAB-02	Inappropriate Payable Modifier	Logical Issue	Informational	Resolved
SAB-03	Lack of Return Value Handling	Logical Issue	Minor	Partially Resolved
SAB-04	Function Should Be Declared External	Gas Optimization	Informational	Resolved
SAB-05	Possible Residue in Current Contract	Logical Issue	Informational	Acknowledged

ID	Title	Category	Severity	Status
SAB-06	Non-Optimal Parameter Set	Logical Issue	Informational	Resolved
SAC-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
SAT-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
SAT-02	Division Before Multiplication	Mathematical Operations	Informational	Resolved
SAT-03	Inappropriate Payable Modifier	Logical Issue	Informational	Resolved
SAT-04	Lack of Return Value Handling	Logical Issue	Minor	Partially Resolved
SAT-05	Non-Optimal Parameter Set	Logical Issue	Informational	Resolved
SCK-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
SCK-02	Lack of Checks for Reentrancy	Logical Issue	Major	Resolved
SLC-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
SLC-02	Inappropriate Payable Modifier	Logical Issue	Informational	Resolved
SLC-03	Lack of Return Value Handling	Logical Issue	Minor	Partially Resolved
SLC-04	Non-Optimal Parameter Set	Logical Issue	Informational	Resolved
SPO-01	Centralization Risks	**Centralization / Privilege**	**Major**	**Partially Resolved**
SPO-02	Mismatch Between Comment and Code	Coding Style	Informational	Resolved
SPO-03	Function Should Be Declared External	Gas Optimization	Informational	Resolved
SVC-01	Centralization Risks	**Centralization / Privilege**	**Major**	**Partially Resolved**
SVC-02	Function Should Be Declared External	Gas Optimization	Informational	Resolved
SWM-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
SWM-02	Inappropriate Payable Modifier	Logical Issue	Informational	Resolved

ID	Title	Category	Severity	Status
SWM-03	Lack of Return Value Handling	Logical Issue	Minor	Partially Resolved
SWM-04	Non-Optimal Parameter Set	Logical Issue	Informational	Resolved
TCK-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved
TCK-02	Lack of Checks for Reentrancy	Logical Issue	Minor	Resolved
VCK-01	Potential Liquidating Issue	Logical Issue	Informational	Resolved
VCK-02	Unexpected Revert	Logical Issue	Minor	Resolved
VCK-03	Risk When Opening a Farming Position	Logical Issue	Major	Resolved
VCK-04	Residue in the Contract	Logical Issue	Minor	Resolved
VCK-05	Function Should Be Declared External	Gas Optimization	Informational	Resolved
WCC-01	Unkillable Position When Worker Is Unstable	Logical Issue	Minor	Resolved
WCC-02	Boolean Function Never Returns False	Logical Issue	Informational	Resolved
WCC-03	Centralization Risks I	**Centralization / Privilege**	**Major**	**Partially Resolved**
WCC-04	Centralization Risks II	**Centralization / Privilege**	**Major**	**Partially Resolved**
WCC-05	Function Should Be Declared External	Gas Optimization	Informational	Resolved
WNR-01	Function Should Be Declared External	Gas Optimization	Informational	Resolved

ATC-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	token/AlpacaToken.sol: 32, 42, 61, 65, 69, 73, 77, 112, 124	Resolved

Description

The functions which are never called internally within the contract should have external visibility. For example:

- `setReleaseBlock`
- `unlockedSupply`
- `burn`
- `totalBalanceOf`
- `lockOf`
- `lastUnlockBlock`
- `lock`
- `unlock`
- `transferAll`

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

ATC-02 | Unexpected Token Locking

Category	Severity	Location	Status
Logical Issue	Informational	token/AlpacaToken.sol: 93, 124	Resolved

Description

The function `transferAll` in L124 is designed to transfer all the Alpaca tokens from `msg.sender` to the input address `_to` (including the locked ones), and meanwhile, it would update `_lastUnlockBlock[_to]` to be the max among `_lastUnlockBlock[_to]`, `startReleaseBlock` and `_lastUnlockBlock[msg.sender]`:

```
124   function transferAll(address _to) public {
125     _locks[_to] = _locks[_to].add(_locks[msg.sender]);
126
127     if (_lastUnlockBlock[_to] < startReleaseBlock) {
128       _lastUnlockBlock[_to] = startReleaseBlock;
129     }
130
131     if (_lastUnlockBlock[_to] < _lastUnlockBlock[msg.sender]) {
132       _lastUnlockBlock[_to] = _lastUnlockBlock[msg.sender];
133     }
134
135     _locks[msg.sender] = 0;
136     _lastUnlockBlock[msg.sender] = 0;
137
138     _transfer(msg.sender, _to, balanceOf(msg.sender));
139   }
```

The mapping `_lastUnlockBlock` is used to calculate the percentage of the tokens that each user can unlock. According to the calculation in the function `canUnlockAmount` in L106~107, the percentage of the tokens that can be unlocked would decrease when `_lastUnlockBlock[_account]` increases:

```
93    function canUnlockAmount(address _account) public view returns (uint256) {
94        ...
95        uint256 releasedBlock = block.number.sub(_lastUnlockBlock[_account]);
96        uint256 blockLeft = endReleaseBlock.sub(_lastUnlockBlock[_account]);
97        return _locks[_account].mul(releasedBlock).div(blockLeft);
98        ...
99    }
```

Combining the logic described above, by calling the function `transferAll`, a malicious user can increase the `_lastUnlockBlock` of another user to decrease the percentage of the tokens that the second user can

unlock. For instance,

- Assume the current state is: `startReleaseBlock = 900`, `endReleaseBlock = 1050` and `block.number = 1040`. User Alice has 100 tokens and `_lastUnlockBlock[Alice] = 1000`.
- In normal case, Alice can unlock 80 tokens (100 * (1040 - 1000) / (1050 - 1000)) per logic implementation in `canUnlockAmount`.
- Malicious user Bob can prevent her from unlocking so many tokens by calling `transferAll(Alice)` if his status is `_lastUnlockBlock[Bob] = 1040`. After calling the function, the state of Alice changes to `_lastUnlockBlock[Alice] = 1040`.
- As a result, Alice can only unlock 0 token (100*(1040 - 1040) / (1050 - 1000)) at the moment.

Recommendation

We recommend the team review the design and ensure this is an intended design.

Alleviation

(**Alpaca Team Response**) It is there for saving user's funds when their account gets hacked, so we have a way to transfer their assets to their new wallet. Our lockup period has been passed. And none of our users get attacked by this function.

(**CertiK**) We agree it will be back to normal when lockup period is passed. We still suggest the team should be cautious about this potential issue which might influence user experience.

ATC-03 | Lack of State Update in `manualMint`

Category	Severity	Location	Status
Logical Issue	Critical	token/AlpacaToken.sol: 50, 15	Resolved

Description

According to the naming pattern of the variable `manualMinted` declared in L15 in the contract `AlpacaToken`, we assume it should be used to record the amount of the tokens that are minted manually.

```
15    uint256 public manualMinted = 0;
```

The `require` check in L51 is supposed to restrain the owner by checking whether the amount of manually minted tokens exceeds the limitation.

```
50    function manualMint(address _to, uint256 _amount) public onlyOwner {
51      require(manualMinted <= manualMintLimit, "mint limit exceeded");
52      mint(_to, _amount);
53    }
```

However, the function `manualMint` does not update the value of `manualMinted` after calling `mint`. Therefore, the owner can manually mint tokens without any restriction since `manualMinted` will always remain 0. Although the contract `Shield` sets a limit on the amount of manually minted Alpaca tokens, the `AlpacaToken` contract itself is vulnerable.

Recommendation

We recommend updating the state `manualMinted` whenever the function `manualMint` is called, to ensure the tokens manually minted are under control.

Alleviation

(**Alpaca Team Response**) This is a known issue and fixed through Shield contract.

(**CertiK**) Please ensure the contract `AlpacaToken` will always bundle with its owner contract together which sets up the limitation properly, since as an independent contract, the `AlpacaToken` will allow the owner to mint an unlimited amount of tokens, thus being vulnerable alone.

ATC-04 | Centralization Risks I

Category	Severity	Location	Status
Centralization / Privilege	Major	token/AlpacaToken.sol: 50	Partially Resolved

Description

The public facing function `manualMint` allows the owner to mint the Alpaca tokens for a certain account:

```
50    function manualMint(address _to, uint256 _amount) public onlyOwner {
51      require(manualMinted <= manualMintLimit, "mint limit exceeded");
52      mint(_to, _amount);
53    }
```

Our concern is, if the owner accidentally and improperly mints the Alpaca tokens, the price of the Alpaca token would be influenced and thus the users/project would suffer unexpected losses.

Recommendation

We recommend the team review the design and ensure minimum centralization risk.

Alleviation

(**Alpaca Team Response**) `ManualMint` function needs to be called from the `Shield` contract. And it already has a manualMint capped (8m for Warchest portion). Hence, it is already self-prevent as there is only 8m ALPACA that can be manually minted. So, the team must think carefully about when minting this portion. Plus, `Shield` contract is also owned by `Timelock`. Hence, if there is a warchest mint without specific reason, token holders can just dump ALPACA to exit all their positions within 24 hours.

(**CertiK**) Setting up an upper bound does not prevent abusively minting tokens within the range. The team should be careful about using the `manualMint` function. Besides, please ensure the proper setup of the owner role. Meanwhile, to improve the trustworthiness of the project, any dynamic runtime update in the project should be notified to the community in advance.

ATC-05 | Centralization Risks II

Category	Severity	Location	Status
Centralization / Privilege	Major	token/AlpacaToken.sol: 32, 77	Resolved

Description

The function `setReleaseBlock` in L32 allows the owner to modify the significant state variables `startReleaseBlock` and `endReleaseBlock`, which determine the amount of the tokens that the user can `unlock` per the implementation of the contract:

```
32    function setReleaseBlock(uint256 _startReleaseBlock, uint256 _endReleaseBlock)
public onlyOwner {
33      require(_endReleaseBlock > _startReleaseBlock, "endReleaseBlock <
startReleaseBlock");
34      startReleaseBlock = _startReleaseBlock;
35      endReleaseBlock = _endReleaseBlock;
36    }
```

Meanwhile, the function `lock` allows the owner to lock a user's tokens until `startRleaseBlock`:

```
77  function lock(address _account, uint256 _amount) public onlyOwner {
78      ...
79      _transfer(_account, address(this), _amount);
80
81      _locks[_account] = _locks[_account].add(_amount);
82      _totalLock = _totalLock.add(_amount);
83
84      if (_lastUnlockBlock[_account] < startReleaseBlock) {
85        _lastUnlockBlock[_account] = startReleaseBlock;
86      }
87      ...
88    }
```

Our concern is, if the owner accidentally and improperly calls the function `setReleaseBlock` to modify the state `startReleaseBlock,` and then calls the function `lock` to lock a user's tokens, it might lead to the result that the user cannot withdraw his/her assets on time, thus introducing centralization risks.

Recommendation

We recommend the team review the design and ensure minimum centralization risk.

Alleviation

(**Alpaca Team Response**) Please note that this function is not accessible anymore on production as the owner of AlpacaToken is a FairLaunch contract, and the FairLaunch contract doesn't have the functionality to access this function.

(**CertiK**) Please ensure the contract `AlpacaToken` will always bundle with the contract `FairLaunch` together.

ATC-06 | Unused Function `setReleaseBlock`

Category	Severity	Location	Status
Gas Optimization	Informational	token/AlpacaToken.sol: 32	Resolved

Description

The function `setReleaseBlock` in L32 can only be called by the owner, which is the contract `FairLaunch` according to the project logic:

```
32  function setReleaseBlock(uint256 _startReleaseBlock, uint256 _endReleaseBlock) public
onlyOwner
```

However, in the contract `FairLaunch`, there is no function calling `AlpacaToken.setReleaseBlock`. Hence, the function `startReleaseBlock` can be safely omitted.

Recommendation

We recommend removing the function `startReleaseBlock` in the aforementioned line.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7744419477522dad110205aed6809f80895e4fd0.

ATC-07 | Variables Should Be Declared Constant

Category	Severity	Location	Status
Gas Optimization	Informational	token/AlpacaToken.sol: 9, 14	Resolved

Description

The state variables `_cap` and `manualMintLimit` do not change within the contract and thus should be declared constant for gas saving.

Recommendation

We recommend adding the constant attributes to the aforementioned variables.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

CIV-01 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	protocol/ConfigurableInterestVaultConfig.sol: 49, 68	Partially Resolved

Description

The function `setParams` in L49 allows the owner to change important configurations of the contract after contract initialization:

```
49   function setParams(
50     uint256 _minDebtSize,
51     uint256 _reservePoolBps,
52     uint256 _killBps,
53     InterestModel _interestModel,
54     address _wrappedNative,
55     address _wNativeRelayer,
56     address _fairLaunch
57   ) public onlyOwner {
58     minDebtSize = _minDebtSize;
59     getReservePoolBps = _reservePoolBps;
60     getKillBps = _killBps;
61     interestModel = _interestModel;
62     wrappedNative = _wrappedNative;
63     wNativeRelayer = _wNativeRelayer;
64     fairLaunch = _fairLaunch;
65   }
```

These configuration parameters are of great significance to the contract and would directly influence the income of both the users and the project. For instance, `_killBps` is a critical parameter to calculate the reward before killing a position. If it is accidentally and improperly modified, the reward might not be calculated correctly, and thus the users and project might suffer unexpected loss.

Similarly, the function `setWorkers` updates `workers` with configuration parameters:

```
68   function setWorkers(address[] calldata addrs, IWorkerConfig[] calldata configs)
external onlyOwner {
69     ...
70     for (uint256 idx = 0; idx < addrs.length; idx++) {
71       workers[addrs[idx]] = configs[idx];
72     }
73   }
```

The state `workers` in the contract also performs a critical role in executing the core logic like `Vault. work`. The configuration parameters bundled with the `workers` would influence the behavior of the contract. Our concern is if the owner accidentally updates the significant configurations, it would influence the entire project logic, which might cause some unexpected loss.

Recommendation

We recommend the team review the design and ensure minimum centralization risk.

Alleviation

(**Alpaca Team Response**) All ConfigurableInterestVaultConfigs are owned by a Timelock contract with 24 hours delay. Hence, tx that will trigger `setParams` and `setWorkers` need to be queued 24 hours in advance. So, if there is a malicious attempt from us, everyone has 24 hours to exit everything.

(**CertiK**) We agree with the solution above. We recommend the team set up the owner role properly. Meanwhile, to improve the trustworthiness of the project, any dynamic runtime update in the project should be notified to the community in advance.

CIV-02 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/ConfigurableInterestVaultConfig.sol: 30	Resolved

Description

The function `initialize` is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

DTC-01 | Lack of Allowance Check In `transferFrom`

Category	Severity	Location	Status
Logical Issue	Critical	protocol/DebtToken.sol: 54~59	Resolved

Description

The public facing function `transferFrom` in L54 can transfer any desired amount of tokens from the input address `from` to the input address `to`, whenever these two addresses are both approved holders:

```
  54    function transferFrom(address from, address to, uint256 amount) public override
returns (bool) {
  55      require(okHolders[from], "debtToken::transferFrom:: unapproved holder in from");
  56      require(okHolders[to], "debtToken::transferFrom:: unapproved holder in to");
  57      _transfer(from, to, amount);
  58      return true;
  59    }
```

The only checks in the function `transferFrom` above are the `require` statements in L55~56, checking if the sender and the receiver are both approved. Without allowance check before token transferring, a malicious approved holder A can drain the tokens from another approved holder B by calling:

```
    transferFrom(B, A, amount);
```

Since no allowance check is applied within the function `transferFrom`, the transaction above can be executed as long as B has enough tokens.

Recommendation

We recommend adding an allowance check in the function `transferFrom` as in openzeppelin.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

(**CertiK**) Please ensure the `sub` function in the update commit will revert on failure.

```
    _approve(from, _msgSender(), allowance(from, _msgSender()).sub(amount, "BEP20:
transfer amount exceeds allowance"));
```

DTC-02 | Dead Code

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/DebtToken.sol: 11, 15	Acknowledged

Description

The variable `timelock` and the modifier `onlyTimelock` defined in the aforementioned lines are not playing any actual role in the contract, and thus can be safely omitted.

Recommendation

We recommend removing the dead code if it is not used anywhere.

Alleviation

(**Alpaca Team Response**) `onlyTimelock` is reserved for the future used as DebtToken is an upgradable contract. So, we would like to reserve memory allocation for a Timelock variable since day 1.

DTC-03 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/DebtToken.sol: 20	Resolved

Description

The function `initialize` is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

FLC-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	token/FairLaunch.sol: 100, 106, 119, 143, 174, 242, 258, 262, 284, 305	Resolved

Description

The functions which are never called internally within the contract should have external visibility. For example:

- `setAlpacaPerBlock`
- `setBonus`
- `addPool`
- `setPool`
- `manualMint`
- `deposit`
- `withdraw`
- `withdrawAll`
- `harvest`
- `emergencyWithdraw`.

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

FLC-02 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	Minor	token/FairLaunch.sol: 320, 322	Resolved

Description

According to the standard `IERC20` interface (which `AlpacaToken` inherits from), the function `transfer` is not a void-returning function. However, in `FairLaunch` contract, the return value of the function `transfer` is not handled properly:

```
319     if (_amount > alpacaBal) {
320       alpaca.transfer(_to, alpacaBal);
321     } else {
322       alpaca.transfer(_to, _amount);
323     }
```

Ignoring the return value of the function `transfer` might cause some unexpected exceptions, especially if the called function doesn't revert automatically on failure.

Recommendation

We recommend checking the output of the aforementioned function, and continuing processing when receiving a proper returned value, otherwise reverting.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

FLC-03 | Lack of Checks for Reentrancy

Category	Severity	Location	Status
Logical Issue	Major	token/FairLaunch.sol: 119, 143, 213, 242, 258, 262, 284, 305	Partially Resolved

Description

The functions that contain state update(s) after external call(s) are potentially vulnerable to reentrancy attack. For example:

- `addPool`
- `setPool`
- `updatePool`
- `deposit`
- `withdraw`
- `withdrawAll`
- `harvest`
- `emergencyWithdraw`

These functions should apply reentrancy guard rails.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

The development team heeded our advice and partially resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26, and promised to fix the rest in the future.

FLC-04 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	Informational	token/FairLaunch.sol: 198~199, 230, 235	Partially Resolved

Description

The mathematical operations in the aforementioned lines perform divisions before multiplications. In L198~199, it divides `totalAllocPoint` before multiplying `1e12`, and in L230, it divides `10` before multiplying `bonusLockUpBps.` It is highly recommended to perform multiplication before division to avoid potential loss of precision.

```
198  uint256 alpacaReward =
multiplier.mul(alpacaPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
199  accAlpacaPerShare = accAlpacaPerShare.add(alpacaReward.mul(1e12).div(lpSupply));
```

```
230  alpaca.lock(devaddr, alpacaReward.div(10).mul(bonusLockUpBps).div(10000));
```

Recommendation

We recommend applying multiplications before divisions if integer overflow would not happen. Then the L198~199 and L230 can be updated as below.

```
198  accAlpacaPerShare =
accAlpacaPerShare.add(multiplier.mul(alpacaPerBlock).mul(pool.allocPoint).mul(1e12).div(t
otalAllocPoint).div(lpSupply));
```

```
230  alpaca.lock(devaddr, alpacaReward.mul(bonusLockUpBps).div(10).div(10000));
```

Alleviation

The development team partially heeded our advice and partially resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26

FLV-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	token/FairLaunchV2.sol: 86, 96, 114, 137, 199, 221, 242, 280	Resolved

Description

The functions which are never called internally within the contract should have external visibility. For example:

- `init`
- `poolLength`
- `addPool`
- `setPool`
- `deposit`
- `withdraw`
- `harvest`
- `emergencyWithdraw`

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

FLV-02 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	Minor	token/FairLaunchV2.sol: 90, 266, 269	Resolved

Description

The function `approve` is not a void-returning function per the `IERC20` interface. In the aforementioned lines, the return value of the function `approve` is not handled properly, for instance:

```
90  dummyToken.approve(address(FAIR_LAUNCH_V1), balance);
```

Ignoring the return value of the function `approve` might cause some unexpected exceptions, especially if the called function doesn't revert automatically on failure.

Recommendation

We recommend checking the output of the aforementioned function, and continuing processing when receiving a proper returned value, otherwise reverting.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

FLV-03 | Lack of Checks for Reentrancy

Category	Severity	Location	Status
Logical Issue	Major	token/FairLaunchV2.sol: 86, 137, 180, 199, 221, 242, 280	Partially Resolved

Description

The functions that contain state update(s) after the external call(s) are potentially vulnerable to reentrancy attack. For example

- `init`
- `setPool`
- `updatePool`
- `deposit`
- `withdraw`
- `harvest`
- `emergencyWithdraw`

These functions should apply reentrancy guard rails.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

The development team heeded our advice and partially resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26, and promised to fix the rest in the future.

FLV-04 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	Informational	token/FairLaunchV2.sol: 155~156, 186~187	Partially Resolved

Description

The mathematical operations in the aforementioned lines perform divisions before multiplications. In L155~156, it divides `totalAllocPoint` before multiplying `ACC_ALPACA_PRECISION`. It is highly recommended to perform multiplication before division to avoid potential loss of precision.

```
155         uint256 alpacaReward = blocks.mul(alpacaPerBlock()).mul(pool.allocPoint) /
totalAllocPoint;
156         accAlpacaPerShare =
accAlpacaPerShare.add(alpacaReward.mul(ACC_ALPACA_PRECISION) / lpSupply);
```

Recommendation

We recommend applying multiplications before divisions if the integer overflow would not happen. Then the L155~156 can be updated as below

```
      accAlpacaPerShare =
accAlpacaPerShare.add(blocks.mul(alpacaPerBlock()).mul(pool.allocPoint).mul(ACC_ALPACA_PR
ECISION).div(totalAllocPoint).div(lpSupply));
```

Alleviation

The development team partially heeded our advice and partially resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26

ITR-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/IbTokenRouter.sol: 23, 275, 320	Resolved

Description

The functions `initialize`, `removeLiquidityToken` and `removeLiquidityAllAlpaca` in the aforementioned lines are never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

LRC-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	token/lockers/LinearRelease.sol: 48, 52, 56, 95	Resolved

Description

The functions which are never called internally within the contract should have external visibility. For example:

- `calLockAmount`
- `lockOf`
- `lock`
- `claim`

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

LRC-02 | Redundant Data Structure

Category	Severity	Location	Status
Gas Optimization	Informational	token/lockers/LinearRelease.sol: 69~93	Resolved

Description

The arrays `_rewardTokens[]` and `_rewardAmounts[]` declared in L88 and L90 are single element arrays (only `_rewardTokens[0]` and `_rewardAmounts[0]` are declared and used):

```
88      IERC20[] memory _rewardTokens = new IERC20[](1);
89      _rewardTokens[0] = (token);
90      uint256[] memory _rewardAmounts = new uint256[](1);
91      _rewardAmounts[0] = amount;
```

Therefore, the arrays above can be replaced with single value variables `_rewardTokens` and `_rewardAmounts`, or use `token` and `amount` instead.

Recommendation

We recommend replacing `_rewardTokens[]` and `_rewardAmounts[]` with `token` and `amount`.

Alleviation

(**Alpaca Team Response**) This is reserved for the case that the contract that is implemented ILocker distributes more than 1 reward for users.

PWC-01 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	Minor	protocol/workers/PancakeswapWorker.sol: 147, 174, 219	Partially Resolved

Description

According to `IUniswapV2Router02` and `IERC20` interfaces, the functions `swapExactTokensForTokens` and `transfer` are not void-returning functions. However, in this contract, the return values of the functions are not handled properly:

```
147     router.swapExactTokensForTokens(reward.sub(bounty), 0, path, address(this),
now);
```

```
174     lpToken.transfer(address(liqStrat), lpToken.balanceOf(address(this)));
```

```
219     lpToken.transfer(address(liqStrat), lpToken.balanceOf(address(this)));
```

Ignoring the return values of these functions might cause some unexpected exceptions, especially if the called functions don't revert automatically on failure.

Recommendation

We recommend checking the output values of the aforementioned functions, and continuing processing when receiving proper returned values, otherwise reverting.

Alleviation

The development team heeded our advice and handled the return value of `transfer` in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26, and decided to leave `swapExactTokensForTokens` as it was.

(**CertiK**) Per the current **PancakeRouter** contract design, the function `swapExactTokensForTokens` will automatically revert on failure, which is safe. However, we encourage the team to be cautious about any modification or update of **PancakeRouter** to ensure these external calls coordinate well with your project logic.

PWC-02 | Lack of Event for Significant Transaction

Category	Severity	Location	Status
Data Flow	Informational	protocol/workers/PancakeswapWorker.sol: 164	Resolved

Description

The function `work` in L164 performs a significant role in the contract. Therefore, logging this action is highly recommended.

Recommendation

We recommend emitting an event in the `work` function, as what is done in the `reinvest` and `liquidate` functions.

Alleviation

(**Alpaca Team Response**) There are events emitted in `_addShare()` and `_removeShare()` functions already. Hence, there is no need to emit events in `work()`.

PWC-03 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/workers/PancakeswapWorker.sol	Resolved

Description

The functions `initialize` and `reinvest` are never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

PWC-04 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	protocol/workers/PancakeswapWorker.sol: 295	Partially Resolved

Description

The function `setCriticalStrategies` in L295 allows the owner to change the execution strategies `addStrat` and `liqStrat`, after contract initialization, and thus could potentially influence the contract execution logic in an improper way. For example, if the owner accidentally updates the strategies to some vulnerable contracts, tokens from the current contract might be stolen away:

```
  function setCriticalStrategies(IStrategy _addStrat, IStrategy _liqStrat) external
onlyOwner {
    addStrat = _addStrat;
    liqStrat = _liqStrat;
  }
```

Recommendation

We recommend the team review the design and ensure minimum centralization risk.

Alleviation

(**Alpaca Team Response**) All workers are owned by a Timelock contract with 24 hours delay. Hence, tx that will trigger `setCriticalStrategies` needs to be queued 24 hours in advance. So, if there is a malicious attempt from us, everyone has 24 hours to exit everything.

(**CertiK**) We agree with the solution above. We recommend the team set up the owner role properly. Meanwhile, to improve the trustworthiness of the project, any dynamic runtime update in the project should be notified to the community in advance.

PWC-05 | Non-Optimal Parameters Passed to Strategy

Category	Severity	Location	Status
Logical Issue	Minor	protocol/workers/PancakeswapWorker.sol: 150, 220	Partially Resolved

Description

In the function `reinvest` of the current contract, `addStrat` is executed in L150, where the last `0` (encoded in `abi.encode(baseToken, quoteToken, 0)`) would be passed down as the value of the parameter `minLPAmount` in the `execute` function of the strategy contracts:

```
150     addStrat.execute(address(0), 0, abi.encode(baseToken, quoteToken, 0));
```

For example, in the `execute` function of the strategy contract `StrategyAddBaseTokenOnly`:

```
69      require(moreLPAmount >= minLPAmount, "StrategyAddBaseTokenOnly::execute::
insufficient LP tokens received");
```

The `require` statement is used as a guard rail to ensure the minimum amount of LP tokens should be received after adding liquidity. However, in this case (when setting `minLPAmount` as 0), the above `require` statement will always be passed, since `moreLPAmount` will always be non-negative. Thus the above require statement is not playing an effective role. Our concern is it might be vulnerable to front running attack.

Similarly, in the function `liquidate`, `liqStrat` is executed in L220, where the last `0` (encoded in `abi.encode(baseToken, quoteToken, 0)` would be the value of parameter `minBaseToken` in the `execute` function of the strategy contracts:

```
220     liqStrat.execute(address(0), 0, abi.encode(baseToken, quoteToken, 0));
```

For example, in the `execute` function of the strategy contract `StrategyLiquidate`:

```
56      require(balance >= minBaseToken, "StrategyLiquidate::execute:: insufficient
baseToken received");
```

It is used as a guard rail to ensure the minimum amount of `baseToken` balance after removing liquidity and token swap. However, in this case (`minBaseToken` being 0), it will not behave as an effective guard rail. It

would make no difference with or without this `require` check when the `minBaseToken` is set as 0 since the `balance` will always be non-negative. Hence, it might be vulnerable to front running attack.

Recommendation

We recommend carefully setting the last parameter in L150, corresponding to `minLPAmount` in `addStrat`, and the last parameter in L220, corresponding to `minBaseToken` in `liquidate`, as some better evaluated non-zero values, to improve the safety overall.

Alleviation

(**Alpaca Team Response**) For `reinvest`, the function is triggered by our reinvest bot and we run it every 30 mins. From running the platform for 2 months, the total amount of `$CAKE` that is earned during 30 mins compared to `$CAKE` liquidity is quite small. For `liquidate`, we have to liquidate positions as fast as possible. This is due to the position already in the killing zone. Passing `minBaseToken` there would cause transaction revert and not be able to liquidate the position before it is going underwater.

(**CertiK**) We agree the risk is relatively low for small transactions. We still suggest the team should be cautious about the potential attack for big transactions and carefully set the value of parameters `minLPAmount` and `minBaseToken` (to be encoded in `data`) whenever calling the strategy `execute` function.

SAB-01 | Unused Variable

Category	Severity	Location	Status
Dead Code	Informational	protocol/strategies/pancakeswap/StrategyAddBaseTokenOnly.sol: 19	Resolved

Description

In L19, the contract declares a variable `wNative`, The variable is not used within the contract and thus can be safely omitted.

```
19    address public wNative;
```

Recommendation

We recommend removing the variable in the aforementioned line.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SAB-02 | Inappropriate Payable Modifier

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyAddBaseTokenOnly.sol: 38	Resolved

Description

In this contract, there is no such payable `receive()` or `fallback()`, which means the contract itself cannot receive any native token (BNB or ETH). However, the function `execute()` (L35) comes with the payable modifier, which is conflict with its design intention.

```
35  function execute(address /* user */, uint256 /* debt */, bytes calldata data)
36    external
37    override
38    payable
39    nonReentrant
40  {
41    ...
```

Recommendation

We recommend removing the `payable` modifier from the function `execute` in the aforementioned contract.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SAB-03 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	Minor	protocol/strategies/pancakeswap/StrategyAddBaseTokenOnly.sol: 64 ~70	Partially Resolved

Description

The functions `swapExactTokensForTokens` and `transfer` in the aforementioned lines are not void-returning functions per `IUniswapV2Router02` and `IERC20` interfaces. Ignoring the return values of these functions might cause some unexpected exceptions, especially if the called functions don't revert automatically on failure.

In the `StrategyAddBaseTokenOnly` contract, the return values of the functions are not handled properly:

```
64      router.swapExactTokensForTokens(aIn, 0, path, address(this), now);
```

```
70      lpToken.transfer(msg.sender, lpToken.balanceOf(address(this)));
```

Recommendation

We recommend checking the output of the aforementioned functions, and continuing processing when receiving proper returned values, otherwise reverting.

Alleviation

The development team heeded our advice and handled the return value of `transfer` in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26, and decided to leave `swapExactTokensForTokens` as it was.

[**CertiK**] In the current **PancakeRouter** contract design, the function `swapExactTokensForTokens` will automatically revert on failure, which is safe. However, we encourage the team to be cautious about any modification or update of the **PancakeRouter** to ensure the external calls coordinate well with your project logic.

SAB-04 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/strategies/pancakeswap/StrategyAddBaseTokenOnly.sol: 26	Resolved

Description

The function `initialize` is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external` for gas optimization.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SAB-05 | Possible Residue in Current Contract

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyAddBaseTokenOnly.sol : 66~68	Acknowledged

Description

The `execute` function in the current contract is to add liquidity to the `Uniswap` liquidity pool by executing L66:

```
66      (,, uint256 moreLPAmount) = router.addLiquidity(
67        baseToken, farmingToken, baseToken.myBalance(), farmingToken.myBalance(), 0,
0, address(this), now
68      );
```

After calling `addLiquidity`, there might be some "baseToken" or "farmingToken" leftover in this strategy contract, because the `addLiquidity` function cannot guarantee all the tokens are sent to the router. That is why function `addLiquidity` has return values telling how many tokens were actually sent. With the current code implementation, the leftover tokens are not returned to the `msg. sender`, but stay in this contract. Our concern is that the leftover tokens might be taken use of by the adjacent next `execute` function caller.

For example:

- Alice is a worker, who would call function `work` in the contract `Vault`. Per the function call chain implemented in the project, the "baseToken" would be transferred from the contract `Vault` to the contract `PancakeswapWorker` and finally to the contract `StrategyAddBaseTokenOnly`.
- After the `execute` function being executed, there might be some "baseToken" leftover, and the leftover "baseToken" would stay in the contract.
- Bob is an attacker who calls the `execute` function right after Alice finishes her `work`. As a result, he would make use of the leftover "baseToken" (because the contract just cares about the balance `baseToken.myBalance()`, but doesn't care where/who these tokens are from) and thus collect `lpToken`.

Recommendation

We recommend transferring the token leftover back to the function caller after `addLiquidity`, or set proper role access on top of the `execute` function.

Alleviation

(**Alpaca Team Response**) The leftover on `baseToken` will be minimal and not really worth the gas fee to return back to users. For strategy add base token only, each floor function in the formula can contribute to at most value of 1 diff from the actual value without floor function. Hence, rounding error will <= 2 Wei.

SAB-06 | Non-Optimal Parameter Set

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyAddBaseTokenOnly.sol: 64~68	Resolved

Description

The 2nd input parameter, `amountOutMin` of the function `swapExactTokensForTokens` indicates the desired minimum amount of the tokens that should be swapped. If less than `amountOutMin` is swapped, this function will revert. However, in the current contract, the parameter `amountOutMin` is set as 0:

```
64      router.swapExactTokensForTokens(aIn, 0, path, address(this), now);
```

Moreover, the 5th and 6th input parameters, `amountAMin` and `amountBMin`, of the function `addLiquidity` indicate the minimum amount of the tokens that should be added to the liquidity pool. If less than `amountAMin` and `amountBMin` are added, this function will revert. However, in the current contract, `amountAMin` and `amountBMin` are set as 0:

```
  66      (,, uint256 moreLPAmount) = router.addLiquidity(
  67        baseToken, farmingToken, baseToken.myBalance(), farmingToken.myBalance(), 0,
0, address(this), now
  68      );
```

This will result in instant token-swap/liquidity-add without considering the market price. Therefore, it is vulnerable to front running attack.

Recommendation

We recommend carefully setting the aforementioned parameters: `amountOutMin`, `amountAMin` and `amountAMin` as some non-zero values, to reduce the potential risks.

Alleviation

(**Alpaca Team Response**) The reason that we passed those params as 0 is because for each strategy we only assert the result after swapping only. We have

```
require(moreLPAmount >= minLPAmount, "StrategyAddBaseTokenOnly::execute:: insufficient LP
tokens received");
```

after swapping. With this line, it basically checks if we received enough LPs and if there is a front running attack or sandwich attack happened. Then this line will be reverted as it is not enough LP received.

(**CertiK**) We agreed on the design mentioned above. We still suggest the team should be cautious about the potential attack for big transactions and carefully set the value of parameter `minLPAmount` (encoded in the input `data`) whenever calling the `execute` function.

SAC-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	token/StronkAlpaca.sol: 105	Resolved

Description

The function `getRelayerAddress` in L105 is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SAT-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/strategies/pancakeswap/StrategyAddTwoSidesOptimal.sol: 27	Resolved

Description

The function `initialize` is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external` for gas optimization.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SAT-02 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	Informational	protocol/strategies/pancakeswap/StrategyAddTwoSidesOptimal.sol: 71	Resolved

Description

The mathematical operations in the aforementioned line perform division before multiplication. In L71, it divides `amtB.add(resB)` before `resA`. It is highly recommended to perform multiplication before division to avoid potential loss of precision.

```
71      uint256 c = _c.mul(1000).div(amtB.add(resB)).mul(resA);
```

Recommendation

We recommend applying multiplications before divisions if integer overflow would not happen. Then the L71 can be updated as below

```
71      uint256 c = _c.mul(1000).mul(resA).div(amtB.add(resB));
```

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SAT-03 | Inappropriate Payable Modifier

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyAddTwoSidesOptimal.sol: 84	Resolved

Description

According to the current code implementation, this strategy contract is not designed to hold any native token (BNB or ETH). However, the `execute` function in the contract is a payable function, which is conflict with its design:

```
  84    function execute(address user, uint256, /* debt */ bytes calldata data) external
override payable nonReentrant
```

Recommendation

We recommend removing the `payable` modifier from the function `execute` in the aforementioned contract.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SAT-04 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	Minor	protocol/strategies/pancakeswap/StrategyAddTwoSidesOptimal.sol: 111~117	Partially Resolved

Description

The functions `swapExactTokensForTokens` and `transfer` are not void-returning functions per `IUniswapV2Router02` and `IERC20` interfaces. In the `StrategyAddTwoSidesOptimal` contract, the return values of the functions are not handled properly:

```
111     if (swapAmt > 0) router.swapExactTokensForTokens(swapAmt, 0, path,
address(this), now);
```

```
117     lpToken.transfer(msg.sender, lpToken.balanceOf(address(this)));
```

Ignoring the return values of these functions might cause some unexpected exceptions, especially if the called functions don't revert automatically on failure.

Recommendation

We recommend checking the output of the aforementioned functions, and continuing processing when receiving proper returned values, otherwise reverting.

Alleviation

The development team heeded our advice and handled the return value of `transfer` in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26, and decided to leave `swapExactTokensForTokens` as it was.

(**CertiK**) Per the current **PancakeRouter** contract design, the `swapExactTokensForTokens` will automatically revert on failure, which is safe. However, we encourage the team to be cautious about any modification or update of the **PancakeRouter** to ensure the external calls coordinate well with your project logic.

SAT-05 | Non-Optimal Parameter Set

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyAddTwoSidesOptimal.sol: 111~115	Resolved

Description

The 2nd input parameter, `amountOutMin` of the function `swapExactTokensForTokens` indicates the desired minimum amount of the tokens that should be swapped. If less than `amountOutMin` is swapped, this function will revert. However, in the current contract, the parameter `amountOutMin` is set as 0:

```
111       router.swapExactTokensForTokens(aIn, 0, path, address(this), now);
```

Moreover, the 5th and 6th input parameters, `amountAMin` and `amountBMin`, of the function `addLiquidity` indicate the minimum amount of the tokens that should be added to the liquidity pool. If less than `amountAMin` and `amountBMin` are added, this function will revert. However, in the current contract, `amountAMin` and `amountBMin` are set as 0:

```
113       (,, uint256 moreLPAmount) = router.addLiquidity(
114         baseToken, farmingToken, baseToken.myBalance(), farmingToken.myBalance(), 0,
0, address(this), now
115       );
```

This will result in instant token-swap/liquidity-add without considering the market price. Therefore, it is vulnerable to front running attack.

Recommendation

We recommend carefully setting the aforementioned parameters: `amountOutMin`, `amountAMin` and `amountAMin` as some non-zero values, to reduce the potential risks.

Alleviation

(**Alpaca Team Response**) The reason that we passed those params as 0 is because for each strategy we only assert the result after swapping only. We have

```
require(moreLPAmount >= minLPAmount, "StrategyAddTwoSidesOptimal::execute:: insufficient
LP tokens received");
```

after swapping. With this line, it basically checks if we received enough LPs and if there is a front running attack or sandwich attack happened. Then this line will be reverted as it is not enough LP received.

(**CertiK**) We agreed on the design mentioned above. We still suggest the team should be cautious about the potential attack for big transactions and carefully set the value of parameter `minLPAmount` (encoded in the input `data`) whenever calling the `execute` function.

SCK-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	token/Shield.sol: 29, 38, 46, 57, 66	Resolved

Description

The functions which are never called internally within the contract should have external visibility. For example:

- `setAlpacaPerBlock`
- `setBonus`
- `mintWarchest`
- `addPool`
- `setPool`

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SCK-02 | Lack of Checks for Reentrancy

Category	Severity	Location	Status
Logical Issue	Major	token/Shield.sol: 29, 38, 46, 57, 66	Resolved

Description

The functions that contain state update(s) after external call(s) are potentially vulnerable to reentrancy attack. For example:

- `setAlpacaPerBlock`
- `setBonus`
- `mintWarchest`
- `addPool`
- `setPool`

These functions should apply reentrancy guard rails.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

(**Alpaca Team Response**) All methods in the Shield contract are needed to be executed through Timelock. Hence, there won't be any reentrancy issue.

(**CertiK**) We agree with the solution above. We recommend the team set up the owner of the contract correctly, and set up proper parameters when calling the functions. Meanwhile, to improve the trustworthiness of the project, any dynamic runtime update in the project should be notified to the community in advance.

SLC-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/strategies/pancakeswap/StrategyLiquidate.sol: 22	Resolved

Description

The function `initialize` is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external` for gas optimization.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SLC-02 | Inappropriate Payable Modifier

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyLiquidate.sol	Resolved

Description

According to the current code implementation, this strategy contract is not designed to receive any native token (BNB or ETH). However, the `execute` function in the contract is a payable function, which is conflict with its design:

```
31  function execute(address /* user */, uint256 /* debt */, bytes calldata data)
32    external
33    override
34    payable
35    nonReentrant
```

Recommendation

We recommend removing the `payable` modifier from the function `execute` in the aforementioned contract.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SLC-03 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	Minor	protocol/strategies/pancakeswap/StrategyLiquidate.sol: 45, 48, 53, 59	Partially Resolved

Description

The functions `approve`, `swapExactTokensForTokens` and `removeLiquidity` are not void-returning functions per `IUniswapV2Router02` and `IERC20` interfaces. In the `StrategyLiquidate` contract, the return values of the functions are not handled properly:

```
45      lpToken.approve(address(router), uint256(-1));
```

```
48      router.removeLiquidity(baseToken, farmingToken, lpToken.balanceOf(address(this)),
0, 0, address(this), now);
```

```
53      router.swapExactTokensForTokens(farmingToken.myBalance(), 0, path, address(this),
now);
```

```
59      lpToken.approve(address(router), 0);
```

Ignoring the return values of these functions might cause some unexpected exceptions, especially if the called functions don't revert automatically on failure.

Recommendation

We recommend checking the output of the aforementioned functions, and continuing processing when receiving proper returned values, otherwise reverting.

Alleviation

The development team heeded our advice and handled the return value of `approve` in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26, and decided to leave `swapExactTokensForTokens` and `removeLiquidity` as they were.

(**CertiK**) Per the current **PancakeRouter** contract design, the functions `swapExactTokensForTokens` and `removeLiquidity` will automatically revert on failure, which are safe. However, we encourage the team to be cautious about any modification or update of the **PancakeRouter** to ensure the external calls coordinate well with your project logic.

SLC-04 | Non-Optimal Parameter Set

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyLiquidate.sol	Resolved

Description

The 4th and 5th input parameters, `amountAMin` and `amountBMin`, of the function `removeLiquidity` indicate the minimum amount of tokens that should be removed from the liquidity pool. If less than `amountAMin` and `amountBMin` are removed, this function will revert. However, in the current contract, `amountAMin` and `amountBMin` are set as 0:

```
48      (,, uint256 moreLPAmount) = router.addLiquidity(
49      router.removeLiquidity(baseToken, farmingToken, lpToken.balanceOf(address(this)),
0, 0, address(this), now);
50      );
```

Moreover, the 2nd input parameter, `amountOutMin` of the function `swapExactTokensForTokens` indicates the desired minimum amount of tokens that should be received. If less than `amountOutMin` is received, this function will revert. However, in the current contract the parameter `amountOutMin` is set as 0:

```
53      router.swapExactTokensForTokens(farmingToken.myBalance(), 0, path, address(this),
now);
```

This will result in instant token-swap/liquidity-add without considering the market price. Therefore, it is vulnerable to front running attack.

Recommendation

We recommend carefully setting the aforementioned parameters: `amountOutMin`, `amountAMin` and `amountAMin` as some non-zero values, to reduce the potential risks.

Alleviation

(**Alpaca Team Response**) The reason that we passed that param as 0 is because for each strategy we only assert the result after swapping only. We have

```
require(balance >= minBaseToken, "StrategyLiquidate::execute:: insufficient baseToken
received");
```

after swapping. With this line, it basically checks if we received enough `baseToken` and if there is a front running attack or sandwich attack happened. Then this line will be reverted as it is not enough `baseToken` received.

(**CertiK**) We agreed on the design mentioned above. We still suggest the team should be cautious about the potential attack for big transactions and carefully set the value of parameter `minBaseToken` (encoded in the input `data`) whenever calling the `execute` function.

SPO-01 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	protocol/SimplePriceOracle.sol: 37	Partially Resolved

Description

The external facing function `setPrices` in the aforementioned line allows the feeder to modify the `price` manually and store it in the `PriceData` for further usage in the project:

```
37  function setPrices(
38    address[] calldata token0s,
39    address[] calldata token1s,
40    uint256[] calldata prices
41  )
42    external
43    onlyFeeder
44  {
45      ...
46      store[token0][token1] = PriceData({
47        price: uint192(price),
48        lastUpdate: uint64(now)
49      });
50      ...
51    }
52  }
```

The `price` stored in `PriceData` performs a significant role and directly influences the income of both users and the project. Hence, our concern is, if the feeder accidentally and improperly, calls the function `setPrices` to modify the `price`, it might cause some unexpected loss, thus introducing centralization risks.

Recommendation

We recommend the team review the design and ensure minimum centralization risk.

Alleviation

(**Alpaca Team Response**) We are currently engaged with several oracle providers. We will soon integrate with an oracle provider.

(**CertiK**) Please ensure integrating with a reliable oracle provider and avoid manually calling the function `setPrice`.

SPO-02 | Mismatch Between Comment and Code

Category	Severity	Location	Status
Coding Style	Informational	protocol/SimplePriceOracle.sol: 36~43	Resolved

Description

According to the comment in L36, the function `setPrices` must be called by the owner:

```
36  /// @dev Set the prices of the token token pairs. Must be called by the owner.
```

From the code implementation, however, the modifier in L43 is `onlyFeeder` instead of `onlyOwner`:

```
37  function setPrices(
38    address[] calldata token0s,
39    address[] calldata token1s,
40    uint256[] calldata prices
41  )
42    external
43    onlyFeeder
```

Recommendation

We recommend the team review the code and correct either the comment or the code implementation.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SPO-03 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/SimplePriceOracle.sol: 26, 32	Resolved

Description

The functions `initialize` and `setFeeder` are never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SVC-01 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	protocol/SimpleVaultConfig.sol: 61, 85	Partially Resolved

Description

The function `setParams` in L61 allows the owner to change important configurations of the contract after contract initialization:

```
61  function setParams(
62    uint256 _minDebtSize,
63    uint256 _reservePoolBps,
64    uint256 _killBps,
65    InterestModel _interestModel,
66    address _wrappedNative,
67    address _wNativeRelayer,
68    address _fairLaunch
69  ) public onlyOwner {
70    minDebtSize = _minDebtSize;
71    getReservePoolBps = _reservePoolBps;
72    getKillBps = _killBps;
73    interestModel = _interestModel;
74    wrappedNative = _wrappedNative;
75    wNativeRelayer = _wNativeRelayer;
76    fairLaunch = _fairLaunch;
77  }
```

Those configuration parameters are of great significance to the contract and would directly influence the income of both the users and the project. For instance, `_killBps` is a critical parameter to calculate the reward before killing a position. If it is accidentally and improperly modified, the reward might not be calculated correctly, and thus the users and project might suffer unexpected loss.

Similarly, the function `setWorkers` update `workers` with configuration parameters:

```
85  function setWorker(
86    address worker,
87    bool _isWorker,
88    bool _acceptDebt,
89    uint256 _workFactor,
90    uint256 _killFactor
91  ) public onlyOwner {
92    workers[worker] = WorkerConfig({
93      isWorker: _isWorker,
```

```
94        acceptDebt: _acceptDebt,
95        workFactor: _workFactor,
96        killFactor: _killFactor
97      });
98    }
```

The state `workers` in the contract also perform a critical role in executing the core logic like `Vault. work`, and the configuration parameters bundled with `workers` would influence the behavior of the contract. Our concern is if the owner accidentally updates the significant configurations, it would influence the entire project logic, which might cause some unexpected loss.

Recommendation

We recommend the team review the design and ensure minimum centralization risk.

Alleviation

(**Alpaca Team Response**) All SimpleVaultConfigs are owned by a Timelock contract with 24 hours delay. Hence, tx that will trigger `setParams` and `setWorkers` need to be queued 24 hours in advance. So, if there is a malicious attempt from us, everyone has 24 hours to exit everything.

(**CertiK**) We agree with the solution above. We recommend the team set up the owner of the contract correctly, and set up proper parameters when calling the functions. Meanwhile, to improve the trustworthiness of the project, any dynamic runtime update in the project should be notified to the community in advance.

SVC-02 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/SimpleVaultConfig.sol: 35, 85	Resolved

Description

The functions `initialize` and `setWorker` are never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SWM-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/strategies/pancakeswap/StrategyWithdrawMinimizeTrading.sol: 23	Resolved

Description

The function `initialize` is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external` for gas optimization.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SWM-02 | Inappropriate Payable Modifier

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyWithdrawMinimizeTrading.sol	Resolved

Description

According to the current code implementation, this strategy contract is not designed to receive any native token (BNB or ETH). However, the `execute` function in the contract is a payable function, which is conflict with its design:

```
35     function execute(address user, uint256 debt, bytes calldata data) external
override payable nonReentrant {
```

Recommendation

We recommend removing the `payable` modifier from the function `execute` in the aforementioned contract.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

SWM-03 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	Minor	protocol/strategies/pancakeswap/StrategyWithdrawMinimizeTrading.sol: 44, 47, 56	Partially Resolved

Description

The functions `approve`, `swapTokensForExactTokens` and `removeLiquidity` are not void-returning functions per `IUniswapV2Router02` and `IERC20` interfaces. In the `StrategyWithdrawMinimizeTrading` contract, the return values of the functions are not handled properly:

```
44      lpToken.approve(address(router), uint256(-1));
```

```
47      router.removeLiquidity(baseToken, farmingToken, lpToken.balanceOf(address(this)),
0, 0, address(this), now);
```

```
56        router.swapTokensForExactTokens(remainingDebt, farmingToken.myBalance(), path,
address(this), now);
```

Ignoring the return values of these functions might cause some unexpected exceptions, especially if the called functions don't revert automatically on failure.

Recommendation

We recommend checking the output of the aforementioned functions, and continuing processing when receiving proper returned values, otherwise reverting.

Alleviation

The development team heeded our advice and handled the return value of `approve` in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26, and decided to leave `swapTokensForExactTokens` and `removeLiquidity` as they were.

(**CertiK**) Per the current **PancakeRouter** contract design, the functions `swapTokensForExactTokens` and `removeLiquidity` will automatically revert on failure, which is safe. However, we encourage the team to be cautious about any modification or update of the **PancakeRouter** to ensure the external calls coordinate well with your project logic.

SWM-04 | Non-Optimal Parameter Set

Category	Severity	Location	Status
Logical Issue	Informational	protocol/strategies/pancakeswap/StrategyWithdrawMinimizeTrading.sol: 47~56	Resolved

Description

The 4th and 5th input parameters, `amountAMin` and `amountBMin`, of the function `removeLiquidity` indicate the minimum amount of tokens that should be removed from the liquidity pool. If less than `amountAMin` and `amountBMin` are removed, this function will revert. In the current contract, `amountAMin` and `amountBMin` are set as 0:

```
47      router.removeLiquidity(baseToken, farmingToken, lpToken.balanceOf(address(this)),
0, 0, address(this), now);
```

Moreover, the 2nd input parameter, `amountInMax` of the function `swapTokensForExactTokens` indicates the maximum amount of input tokens that can be required before the transaction reverts. If more than `amountInMax` is required, this function will revert. In the current contract the parameter `amountInMax` is set as account balance:

```
56        router.swapTokensForExactTokens(remainingDebt, farmingToken.myBalance(), path,
address(this), now);
```

Above setting up will result in instant token-swap/liquidity-removal without considering the market price. Therefore, it is vulnerable to front running attack .

Recommendation

We recommend carefully setting the aforementioned parameters: `amountOutMin`, `amountAMin` and `amountAMin` as some non-zero values, to reduce the potential risks.

Alleviation

(**Alpaca Team Response**) We have

```
require(remainingFarmingToken >= minFarmingToken,
"StrategyWithdrawMinimizeTrading::execute:: insufficient quote tokens received");
```

after swapping. With this line, it basically checks if we have enough `remainingFarmingToken` and if there is a front running attack or sandwich attack happened. Then this line will be reverted as it is not enough `remainingFarmingToken` left.

(**CertiK**) We agreed on the design mentioned above. We still suggest the team should be cautious about the potential attack for big transactions and carefully set the value of parameter `minFarmingToken` (encoded in the input `data`) whenever calling the `execute` function.

TCK-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	Timelock.sol: 50, 59, 67, 80, 91, 111	Resolved

Description

The functions which are never called internally within the contract should have external visibility. For example:

- `setDelay`
- `acceptAdmin`
- `setPendingAdmin`
- `queueTransaction`
- `cancelTransaction`
- `executeTransaction`

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

TCK-02 | Lack of Checks for Reentrancy

Category	Severity	Location	Status
Logical Issue	Minor	Timelock.sol: 111	Resolved

Description

The function `executeTransaction` contains state update after external call. Therefore, the function is potentially vulnerable to reentrancy attack and should apply reentrancy guard rails.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned function to prevent reentrancy attack.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

VCK-01 | Potential Liquidating Issue

Category	Severity	Location	Status
Logical Issue	Informational	protocol/Vault.sol: 311	Resolved

Description

The `kill` function in L311 is designed to kill and liquidate the given position when the `killFactor*health` condition is met in L311:

```
311   require(health.mul(killFactor) < debt.mul(10000), "Vault::kill:: can't liquidate");
```

However, if a user wants to withdraw the assets by calling the `kill` function with a certain position `id`, the `require` check in the aforementioned line is likely to revert since the `health` factor might be large enough. Logically, users would expect to be capable of withdrawing all their assets at any time rather than waiting until the position is "unhealthy". Besides, the asset might be locked permanently if the `killFactor*health` condition is never met.

Recommendation

We recommend the team review the logic and ensure this is an intended design.

Alleviation

(**Alpaca Team Response**) It is intended by design. Kill function is used for liquidating unhealthy positions only. If users want to close the position, they will use the `work` function with either `StrategyLiquidate` or `StrategyMinimizeTrading` not `kill` function.

VCK-02 | Unexpected Revert

Category	Severity	Location	Status
Logical Issue	Minor	protocol/Vault.sol: 317	Resolved

Description

According to the implementation of `sub` function, from where the calculation in L317 would revert if `back < prize`:

```
315     uint256 back = SafeToken.myBalance(token).sub(beforeToken);
316     uint256 prize = back.mul(config.getKillBps()).div(10000);
317     uint256 rest = back.sub(prize);
```

In L316, if `config.getKillBps() > 1000`, `prize` will be larger than `back`, and L317 will revert, instead of performing liquidation. In this case, the contract and the users might suffer unexpected loss since it fails to liquidate in time.

Recommendation

We recommend adding check for the variable `price` after the L316 to minimize the unexpected loss.

```
317     if (prize > back) {
318         prize = back;
319     }
```

Alleviation

The development team heeded our advice and resolved this issue by validating the `killBps` setting in `ConfigurableInterestVaultConfig.setParams()` (to ensure `prize < back`) in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

VCK-03 | Risk When Opening a Farming Position

Category	Severity	Location	Status
Logical Issue	Major	protocol/Vault.sol: 236	Resolved

Description

When opening a farming position with the function `work` in this contract, a user could borrow a huge amount of tokens from the vault. The only significant guard rail to prevent this action is in L280, by checking if it is healthy enough:

```
280  require(health.mul(workFactor) >= debt.mul(10000), "Vault::work:: bad work factor");
```

However, the `health` (one factor of the health evaluation) in L278 is highly dependent on PancakeSwap listing prices, which is quite sensitive to any ongoing on-chain DeFi attack (e.g., flash loan attack):

```
278   uint256 health = IWorker(worker).health(id);
```

On the other hand, the `workFactor` (another factor of the health evaluation) in L279 is a manual-set factor that might not be updated in time:

```
279   uint256 workFactor = config.workFactor(worker, debt);
```

As a result, the project might suffer from the following risk scenario: A user opening a position by loaning a huge amount of tokens successfully, and right after that block, it might already trigger the liquidating signal due to some chain state change. But even if `kill` is triggered right away in the next block (if the bot is working frequently enough), the loss could potentially be big enough to erode the PROJECT principal. The problem could be quite severe because there are not enough checks in the first place. To control the risk, for most yield farming contracts, the loan should be no more than about 40%-60% of the user deposit principal and should never be more than 100% (unless it is a flash loan).

Recommendation

We recommend the team review the flow and minimize the potential risk. We would also suggest setting a threshold to limit the amount of loan according to the user's principal in the first place before any further processing.

Alleviation

`PancakeswapWorker.health()` is one factor of the healthy evaluation which we take the asset price from PCS. I agree that `PancakeswapWorker.health()` alone wouldn't be enough to prevent an on-chain DeFi attack through flash-loan, etc. However, you can see that on the next line `config.workFactor()` L279 which call `ConfigurableInterestVaultConfig.workFactor()` and later call `WorkerConfig.workFactor()` where in this `WorkerConfig.workFactor` we checked various conditions whether the Worker is under the manipulation or not. This includes checking if the reserve is consistent and also if the price of that pair is diff from the oracle more than the diff threshold or not. Hence, if the attacker tries to manipulate the price on PCS, the contract will block the `open,` `close,` and `kill` position until the price on PCS becomes within the diff threshold, and LP becomes consistent.

(**CertiK**) We strongly encourage the team to closely monitor the position activities (open/close) and the state of the opened positions to avoid any potential loss.

VCK-04 | Residue in the Contract

Category	Severity	Location	Status
Logical Issue	Minor	protocol/Vault.sol: 201	Resolved

Description

In the function `withdraw`, the requirement in L201 means nobody can withdraw all the shares left, but instead has to leave residue larger than `1e17` shares in the contract:

```
201  require(totalSupply() > 1e17, "Vault::withdraw:: no tiny shares");
```

For instance, if a user deposits `11e17` tokens and withdraws at once, he can only get back `10e17-1` at most and thus suffer from a loss.

Recommendation

We recommend the team review the flow and ensure this is an intended design.

Alleviation

(**Alpaca Team Response**) The 1e17 loss will be covered by the Alpaca team.

VCK-05 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/Vault.sol: 107, 166	Resolved

Description

The functions `initialize` and `positionInfo` are never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

WCC-01 | Unkillable Position When Worker Is Unstable

Category	Severity	Location	Status
Logical Issue	Minor	protocol/workers/WorkerConfig.sol: 86~89	Resolved

Description

The function `killFactor` in L86 returns the kill factor of a certain worker. It will revert when the worker is not stable:

```
  86  function killFactor(address worker, uint256 /* debt */) external override view
returns (uint256) {
  87      require(isStable(worker), "WorkerConfig::killFactor:: !stable");
  88      return uint256(workers[worker].killFactor);
  89  }
```

When it is not stable, the function `kill` in the contract `Vault` will revert since it is calling the function `killFactor`:

```
function kill(uint256 id) external onlyEOA accrue(0) nonReentrant {
    ...
    uint256 killFactor = config.killFactor(pos.worker, debt);
    require(health.mul(killFactor) < debt.mul(10000), "Vault::kill:: can't liquidate");
    ...
  }
```

The concern is when `health` is low, the user might want to kill the position as soon as possible. However, if the worker is not stable, the function `kill` will not be executed successfully. As a result, the project and the user might suffer unexpected loss since the position cannot be liquidated in time.

Recommendation

We recommend the team review the flow and ensure this is an intended design.

Alleviation

(**Alpaca Team Response**) The function `kill` should be blocked as well. This is due to the fact that if `isStable` is reverted, it means that the price of the given pair on DEX is being manipulated by the attacker. This is to prevent attackers that try to manipulate the price to the point where all positions on our protocol get liquidated.

WCC-02 | Boolean Function Never Returns False

Category	Severity	Location	Status
Logical Issue	Informational	protocol/workers/WorkerConfig.sol: 52	Resolved

Description

The function `isStable` in the aforementioned line is designed to determine whether a worker is stable. It would EITHER return `true`, OR revert, but never return false. The visibility of the function is public, thus it might cause some user experience issues if being called from a client, when the user expects `false` but receives an exception.

Recommendation

We recommend the team review the working flow and ensure this is an intended design

Alleviation

(**Alpaca Team Response**) The reason that we made `isStable` returns bool is because we want to clearly stated `require(isStable(), "")` in functions `acceptDebt`, `workFactor`, and `killFactor` to increase our code readability.

WCC-03 | Centralization Risks I

Category	Severity	Location	Status
Centralization / Privilege	Major	protocol/workers/WorkerConfig.sol: 33	Partially Resolved

Description

The function `setOracle` in the aforementioned line allows the owner to change the Oracle within the project after contract initialization:

```
33   function setOracle(PriceOracle _oracle) external onlyOwner {
34     oracle = _oracle;
35   }
```

Since Oracle provides significant price information for contracts to use within the project, it would be risky if a malicious Oracle is applied. Our concern is if the owner accidentally update the Oracle to a malicious one, it will influence the entire project logic, and might cause some unexpected loss.

Recommendation

We recommend the team review the design and ensure minimum centralization risk.

Alleviation

(**Alpaca Team Response**) All WorkerConfigs are owned by a Timelock contract with 24 hours delay. Hence, tx that will trigger `setOracle` needs to be queued 24 hours in advance. So, if there is a malicious attempt from us, everyone has 24 hours to exit everything.

(**CertiK**) We agree with the solution above. We recommend the team set up the owner role properly. Meanwhile, to improve the trustworthiness of the project, any dynamic runtime update in the project should be notified to the community in advance.

WCC-04 | Centralization Risks II

Category	Severity	Location	Status
Centralization / Privilege	Major	protocol/workers/WorkerConfig.sol: 38	Partially Resolved

Description

The function `setConfigs` in L38 allows the contract owner to change significant configurations of the contract after contract initialization:

```
38    function setConfigs(address[] calldata addrs, Config[] calldata configs) external
onlyOwner {
39      uint256 len = addrs.length;
40      require(configs.length == len, "WorkConfig::setConfigs:: bad len");
41      for (uint256 idx = 0; idx < len; idx++) {
42        workers[addrs[idx]] = Config({
43          acceptDebt: configs[idx].acceptDebt,
44          workFactor: configs[idx].workFactor,
45          killFactor: configs[idx].killFactor,
46          maxPriceDiff: configs[idx].maxPriceDiff
47        });
48      }
49    }
```

Configuration parameters `acceptDebt`, `workFactor` and `killFactor` are quite important factors in the functions `kill` and `work` in the contract `Vault,` which directly influence the income of the project. For instance, the `killFactor` could be modified to block anyone else from liquidating a position. Besides, `maxPriceDiff` is a key factor to determine whether a worker is `stable` and if it is improperly set, it would cause the function `kill` and `work` to revert. Our concern is if the owner accidentally updates the significant configurations to some improper values, it might cause some unexpected loss.

Recommendation

We recommend the team review the design and ensure minimum centralization risk.

Alleviation

(**Alpaca Team Response**) All WorkerConfigs are owned by a Timelock contract with 24 hours delay. Hence, tx that will trigger `setConfigs` needs to be queued 24 hours in advance. So, if there is a malicious attempt from us, everyone has 24 hours to exit everything.

(**CertiK**) We agree with the solution above. We recommend the team set up the owner role properly. Meanwhile, improving the project's trustworthiness and dynamic runtime update in the project should be notified to the community in advance.

WCC-05 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/workers/WorkerConfig.sol: 27	Resolved

Description

The function `initialize` is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

WNR-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	Informational	protocol/WNativeRelayer.sol: 27	Resolved

Description

The function `withdraw` is never called internally within the contract and thus should have external visibility.

Recommendation

We recommend changing the visibility of the aforementioned function to `external`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 7b8389ac08f2025af8bad23af0ba7ea91ca94c26.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.