// Smart Contract Security Assessment 11.04.2024 - 11.07.2024

Morpho Vault Strategy
Concrete

HALBORN

Morpho Vault Strategy - Concrete

Prepared by: HALBORN

Last Updated 01/07/2025

Date of Engagement by: November 4th, 2024 - November 7th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	1	0	1	2

TABLE OF CONTENTS

1. Introduction

Concrete engaged Halborn to conduct a security assessment on their smart contracts beginning on November 4th and ending on November 7th, 2024. The security assessment was scoped to the smart contracts provided to the Halborn team.

Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

The team at Halborn assigned a full-time security engineer to assess the security of the smart contracts. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the Concrete team :

- Initialize the protectStrategy variable with the current value passed as a parameter, in both addOrReplaceStrategy and removeStrategy functions.
- In the calculateTieredFee function change the comparison operators to inclusive (<= and >=) to ensure that boundary values are correctly included in the fee calculations.
- Refactor the claimRewardsAndSend function to avoid reverting on failure.
- Remove all unused imports.

3. Test Approach And Methodology

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could lead to arithmetic related vulnerabilities.
- Manual testing by custom scripts.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Static Analysis of security for scoped contract, and imported functions. (Slither).
- Local or public testnet deployment (Foundry, Remix IDE).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: sc_earn-v1

(b) Assessed Commit ID: ecced27

(c) Items in scope:

- src/strategies/StrategyBase.sol
- src/strategies/Morpho/MorphoVaultStrategy.sol
- src/libraries/MultiStrategiesVaultHelper.sol
- The following file from e3c6a006923197230320e266e586e859f8eca344 commit was added to the scope:
- src/libraries/MultiStrategiesVaultHelper.sol

Out-of-Scope: Third party dependencies and economic attacks.

FILES AND REPOSITORY

(a) Repository: sc_hub-and-spokes-libraries

(b) Assessed Commit ID: hf9753a

(c) Items in scope:

- src/libraries/MorphoV1Helper.sol
- src/libraries/TokenHelper.sol

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- dd30e9a
- ccf8d86
- e3c6a00
- e3c6a00

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	1	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INADVERTENT CLEARING OF PROTECTSTRATEGY WHEN MANAGING STRATEGIES	HIGH	SOLVED - 11/13/2024
INCORRECT FEE CALCULATION DUE TO EXCLUSIVE COMPARISON OPERATORS	LOW	SOLVED - 12/04/2024
INCONSISTENT REWARD HARVESTING FLOW	INFORMATIONAL	SOLVED - 11/08/2024
UNUSED IMPORTS AND ERRORS	INFORMATIONAL	PARTIALLY SOLVED - 11/08/2024

7. FINDINGS & TECH DETAILS

7.1 INADVERTENT CLEARING OF PROTECTSTRATEGY WHEN MANAGING STRATEGIES

// HIGH

Description

The `MultiStrategyVaultHelpler` library is designed to manage multiple strategies, including a special `protectStrategy` intended to safeguard assets under specific conditions. The `protectStrategy` variable holds the address of the current protect strategy and is critical for the vault's security. However, due to improper initialization and handling of the `protectStrategy` variable within the strategy management functions, specifically `addOrReplaceStrategy` and `removeStrategy`, the `protectStrategy` can be inadvertently set to `address(0)` under the following circumstances:

- **Adding a Non-Protect Strategy**: When adding a new non-protect strategy to the vault without replacing any existing strategies, the `protectStrategy` may be unintentionally reset to `address(0)`, even though the protect strategy was not involved in the operation.

```
197 function addOrReplaceStrategy(
198     Strategy[] storage strategies,
199     Strategy memory newStrategy_,
200     bool replace_,
201     uint256 index_,
202     address protectStrategy_,
203     IERC20 asset
204 ) public returns (address protectStrategy, IStrategy newStrategyIfc, IS
205     // Calculate total allotments of current strategies
206     uint256 allotmentTotals = 0;
207     uint256 len = strategies.length;
208     for (uint256 i = 0; i < len;) {
209         allotmentTotals += strategies[i].allocation.amount;
210         unchecked {
211             i++;
212         }
213     }
214
215     // Adding or replacing strategy based on `replace_` flag
216     if (replace_) {
217         if (index_ >= len) revert InvalidIndex(index_);
218
219         // Ensure replacing doesn't exceed total allotment limit
220         if (
221             allotmentTotals - strategies[index_].allocation.amount + ne
```

```
                > MAX_BASIS_POINTS
        ) {
            revert AllotmentTotalTooHigh();
        }

        // Replace the strategy at `index_`
        stratToBeReplacedIfc = strategies[index_].strategy;
        protectStrategy_ = removeStrategy(stratToBeReplacedIfc, protect

        strategies[index_] = newStrategy_;
    } else {
        // Ensure adding new strategy doesn't exceed total allotment li
        if (allotmentTotals + newStrategy_.allocation.amount > MAX_BASI
            revert AllotmentTotalTooHigh();
        }

        // Add the new strategy to the array
        strategies.push(newStrategy_);
    }

    // Handle protect strategy assignment if applicable
    if (newStrategy_.strategy.isProtectStrategy()) {
        if (protectStrategy_ != address(0)) revert MultipleProtectStrat
        protectStrategy = address(newStrategy_.strategy);
    }

    // Approve the asset for the new strategy
    asset.forceApprove(address(newStrategy_.strategy), type(uint256).ma

    // Return the address of the new strategy
    newStrategyIfc = newStrategy_.strategy;
}
```

- **Removing a Non-Protect Strategy**: When removing a non-protect strategy from the vault, the `protectStrategy` can also be inadvertently reset to `address(0)`, despite the protect strategy remaining in place.

```
function removeStrategy(IStrategy stratToBeRemoved_, address protectStr
    public
    returns (address protectStrategy)
{
    protectStrategy = protectStrategy_;
    // Check if the strategy has any locked assets that cannot be witha
    if (stratToBeRemoved_.getAvailableAssetsForWithdrawal() != stratToB
        revert StrategyHasLockedAssets(address(stratToBeRemoved_));
```

```
205         }
206
207         // Redeem all assets from the strategy if it has any assets
208         if (stratToBeRemoved_.totalAssets() > 0) {
209             stratToBeRemoved_.redeem(stratToBeRemoved_.balanceOf(address(th
210         }
211
212         // Reset protect strategy if the strategy being removed is the prot
213         if (protectStrategy_ == address(stratToBeRemoved_)) {
214             protectStrategy = address(0);
215         }
216
217         // Reset allowance to zero for the strategy being removed
218         asset.forceApprove(address(stratToBeRemoved_), 0);
219     }
```

In both cases, the vault loses its protect strategy without any explicit action taken to remove or replace it, leading to funds losses.

Proof of Concept

The protect strategy is cleared after setting:

```
function test_protectStrategyInadvertentlyCleared() public {
    (ConcreteMultiStrategyVault newVault, Strategy[] memory strats) = _crea

    Strategy memory unprotectedStrategy = _createMockStrategy(IERC20(addres


    Strategy memory protectedStrategy = Strategy({
        strategy: IStrategy(address(new MockERC4626Protect(IERC20(address(a
        allocation: Allocation({index: 0, amount: 3333})
    });

    //Add a protect strategy
    vm.prank(admin);
        newVault.addStrategy(5, false, protectedStrategy);

    address initialProtectStrategy = newVault.protectStrategy();
    assertNotEq(initialProtectStrategy, address(0),
    "Protect strategy should not be address(0)"
    );

    //Add a non-protect strategy
    vm.prank(admin);
        newVault.addStrategy(6, false, unprotectedStrategy);
```

```
    address protectStrategyAfter = newVault.protectStrategy();
    //Protect strategy should not be address(0) after replacement, indicati
    assertEq(protectStrategyAfter, address(0));
}
```

```
Ran 1 test for test/ConcreteMultiStrategyVault.t.sol:ConcreteMultiStrategyVaultTest
[PASS] test_protectStrategyInadvertentlyCleared() (gas: 7474441)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.40ms (713.92µs CPU time)

Ran 1 test suite in 128.19ms (2.40ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:N/S:U (7.5)

Recommendation

It is recommended to initialize the `protectStrategy` variable with the current value passed as a parameter, in both `addOrReplaceStrategy` and `removeStrategy` functions.

Remediation

SOLVED: The **Concrete team** solved the issue in the specified commit id. The `protectStrategy` variable was initialized as recommended.

Remediation Hash

https://github.com/Blueprint-Finance/sc_earn-v1/commit/dd30e9ad47d895d6b71d74ee3f4e6559849091b9

References

Blueprint-Finance/sc_earn-v1/src/libraries/MultiStrategyVaultHelper.sol#L197-L283

7.2 INCORRECT FEE CALCULATION DUE TO EXCLUSIVE COMPARISON OPERATORS

// LOW

Description

The `calculateTieredFee` function, implemented in the `MultiStrategiesVaultHelper`, is intended to compute a tiered performance fee based on the percentage increase of the `shareValue` over the `highWaterMark`. The fee is determined by comparing the calculated difference against predefined fee tiers specified in `fees.performanceFee`.

```
141 function calculateTieredFee(uint256 shareValue, uint256 highWaterMark,
142     public
143     view
144     returns (uint256 fee)
145 {
146     if (shareValue <= highWaterMark) return 0;
147     // Calculate the percentage difference (diff) between share value c
148     uint256 diff =
149         uint256(shareValue.mulDiv(MAX_BASIS_POINTS, highWaterMark, Math
150 
151     // Loop through performance fee tiers
152     uint256 len = fees.performanceFee.length;
153     if (len == 0) return 0;
154     for (uint256 i = 0; i < len;) {
155         if (diff < fees.performanceFee[i].upperBound && diff > fees.per
156             fee = ((shareValue - highWaterMark) * totalSupply).mulDiv(
157                 fees.performanceFee[i].fee, MAX_BASIS_POINTS * 1e18, Ma
158             );
159             break; // Exit loop once the correct tier is found
160         }
161         unchecked {
162             i++;
163         }
164     }
165 }
```

However, due to the use of exclusive comparison operators (`<` and `>`) in the tier matching logic, the function fails to correctly apply fees when the `diff` is exactly equal to the `lowerBound` or `upperBound` of a tier. This can result in scenarios where no fee is charged when it should be, allowing users to avoid paying fees by manipulating the `shareValue` to fall exactly on a tier boundary.

BVSS

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (3.4)

Recommendation

It is recommended to change the comparison operators to inclusive (<= and >=) to ensure that boundary values are correctly included in the fee calculations.

Remediation

SOLVED: The **Concrete team** solved the issue in the specified commit id. The recommended comparison was introduced.

Remediation Hash

https://github.com/Blueprint-Finance/sc_earn-v1/commit/ccf8d86576ef15b05fead40d74a247e4510b9197

References

Blueprint-Finance/sc_earn-v1/src/libraries/MultiStrategyVaultHelper.sol#L141-L165

7.3 INCONSISTENT REWARD HARVESTING FLOW

// INFORMATIONAL

Description

The `_getRewardsToStrategy` function in the `MorphoVaultStrategy` contract is used as the initial step of the reward harvesting process. This function utilizes the `claimRewardsAndSend` function from the **MorphoV1Helper** library:

```
98  function _getRewardsToStrategy(bytes memory data) internal override {
99      //TODO check the rewards based on the distributor
100     MorphoV1Helper.claimRewardsAndSend(address(0), data, 1, true);
101 }
```

The `claimRewardsAndSend` function decodes the data provided during the reward harvesting flow and reverts if the decoded owner is not equal to msg.sender:

```
281 function claimRewardsAndSend(address owner, bytes memory txDataEncoded,
282     public
283 {
284     (address[] memory urd, bytes[] memory txData) = decodeMorphoV1Proof
285     address token;
286     address account;
287     uint256 claimable;
288     uint256 claimed;
289     bytes32[] memory proof;
290     for (uint256 i = 0; i < urd.length;) {
291         (account, token, claimable, proof) = decodeTransactionDataAndPr
292         if (revertIfReceiverInvalid) {
293             if (account != msg.sender) {
294                 revert Errors.InvalidMorphoRewardsReceiver();
295             }
296         }
297         claimed = IClaimMorphoRewards(urd[i]).claim(account, token, cla
298         // SendFundsModality.SEND_THROUGH = SendFundsModality(uint8(0))
299         if (mode == 0) {
300             if (TokenHelper.attemptSafeTransfer(token, owner, claimed,
301                 emit IClaimMorphoRewards.ClaimedRewards(token, claimed)
302             }
303         }
304         unchecked {
305             i++;
306         }
307
```

```
308         }
        }
```

This behavior differs from the reward harvesting logic in other strategies, such as Radian or Silo, which use a `try/catch` approach to handle failures and avoid reverting the entire transaction if rewards cannot be claimed.

BVSS

AO:A/AC:L/AX:M/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (1.7)

Recommendation

It is recommended to refactor the `claimRewardsAndSend` function and implement a `try/catch` mechanism, similar to the other strategies.

Remediation

SOLVED: The **Concrete team** solved the issue in the specified commit id. The `claimRewardsAndSend` function is called with the boolean value set to `false`, which prevents reverts if the decoded owner address does not match `msg.sender`.

Remediation Hash

https://github.com/Blueprint-Finance/sc_earn-v1/commit/e3c6a006923197230320e266e586e859f8eca344

References

Blueprint-Finance/sc_earn-v1/src/strategies/Morpho/MorphoVaultStrategy.sol#L98-L101
Blueprint-Finance/sc_hub-and-spokes-libraries/src/libraries/MorphoV1Helper.sol#L281-L308

7.4 UNUSED IMPORTS AND ERRORS

// INFORMATIONAL

Description

Throughout the code in scope, there are several instances where the imports and errors, are declared but never used.

In `MorphoVaultStrategy.sol`:

- `import {IStrategy, ReturnedRewards} from "../../interfaces/IStrategy.sol";`

In `TokenHelper.sol`:

- `import {IERC20Metadata} from "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";`

In `StrategyBase.sol`:

- `import {Initializable} from "@openzeppelin/contracts/proxy/utils/Initializable.sol";`

In `MultiStrategiesVaultHelper.sol`:

- `error InvalidFeeRecipient();`
- `error ERC20ApproveFail();`

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

It is recommended to remove all unused imports.

Remediation

PARTIALLY SOLVED: The **Concrete team** partially solved the issue in the specified commit id. The unused imports were removed.

Remediation Hash

https://github.com/Blueprint-Finance/sc_hub-and-spokes-libraries/commit/d268ac1f3bf6af752ca33c71288e6dcf124d1918 https://github.com/Blueprint-Finance/sc_earn-v1/commit/e3c6a006923197230320e266e586e859f8eca344

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

The security team assessed all findings identified by the Slither software and everything was categorised as false positives.

Results

- `MorphoV1Helper.sol`:

```
INFO:Detectors:
5 different versions of Solidity are used:
	- Version constraint >=0.5.0 is used by:
		- lib/morpho-blue/src/interfaces/IIrm.sol#2
		- lib/morpho-blue/src/interfaces/IMorpho.sol#2
		- lib/morpho-blue/src/interfaces/IOracle.sol#2
	- Version constraint ^0.8.0 is used by:
		- lib/morpho-blue/src/libraries/ErrorsLib.sol#2
		- lib/morpho-blue/src/libraries/MarketParamsLib.sol#2
		- lib/morpho-blue/src/libraries/MathLib.sol#2
		- lib/morpho-blue/src/libraries/SharesMathLib.sol#2
		- lib/morpho-blue/src/libraries/UtilsLib.sol#2
		- lib/morpho-blue/src/libraries/periphery/MorphoBalancesLib.sol#2
		- lib/morpho-blue/src/libraries/periphery/MorphoLib.sol#2
		- lib/morpho-blue/src/libraries/periphery/MorphoStorageLib.sol#2
	- Version constraint ^0.8.20 is used by:
		- lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4
		- lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#4
		- lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Permit.sol#4
		- lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#4
		- lib/openzeppelin-contracts/contracts/utils/Address.sol#4
		- lib/openzeppelin-contracts/contracts/utils/math/Math.sol#4
		- lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#5
	- Version constraint ^0.8.24 is used by:
		- lib/sc_hub-and-spokes-libraries/src/libraries/AddressLib.sol#2
		- lib/sc_hub-and-spokes-libraries/src/libraries/Errors.sol#2
		- lib/sc_hub-and-spokes-libraries/src/utils/Constants.sol#2
	- Version constraint 0.8.24 is used by:
		- lib/sc_hub-and-spokes-libraries/src/libraries/MorphoV1Helper.sol#2
		- lib/sc_hub-and-spokes-libraries/src/libraries/TokenHelper.sol#2
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
The following unused import(s) in lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol should be removed:
	-import {IERC20Permit} from "../extensions/IERC20Permit.sol"; (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#7)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-imports
INFO:Slither:lib/sc_hub-and-spokes-libraries/src/libraries/MorphoV1Helper.sol analyzed (25 contracts with 91 detectors), 2 result(s) found
```

- `TokenHelper.sol`:

```
INFO:Detectors:
3 different versions of Solidity are used:
	- Version constraint ^0.8.20 is used by:
		- lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4
		- lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#4
		- lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Permit.sol#4
		- lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#4
		- lib/openzeppelin-contracts/contracts/utils/Address.sol#4
	- Version constraint ^0.8.24 is used by:
		- lib/sc_hub-and-spokes-libraries/src/libraries/Errors.sol#2
	- Version constraint 0.8.24 is used by:
		- lib/sc_hub-and-spokes-libraries/src/libraries/TokenHelper.sol#2
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
The following unused import(s) in lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol should be removed:
	-import {IERC20Permit} from "../extensions/IERC20Permit.sol"; (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#7)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-imports
INFO:Slither:lib/sc_hub-and-spokes-libraries/src/libraries/TokenHelper.sol analyzed (7 contracts with 91 detectors), 2 result(s) found
```

- `StrategyBase.sol`:

```
INFO:Detectors:
StrategyBase.harvestRewards(bytes) (src/strategies/StrategyBase.sol#351-385) ignores return value by TokenHelper.attemptSafeTransfer(address(rewardAddress),_vault,netReward,false) (src/strategies/StrategyBase.sol#376)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
StrategyBase.harvestRewards(bytes) (src/strategies/StrategyBase.sol#351-385) has external calls inside a loop: claimedBalance = rewardAddress.balanceOf(address(this)) (src/strategies/StrategyBase.sol#365)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
4 different versions of Solidity are used:
        - Version constraint ^0.8.20 is used by:
                - lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#4
                - lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/Initializable.sol#4
                - lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#4
                - lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/extensions/ERC4626Upgradeable.sol#4
                - lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#4
                - lib/openzeppelin-contracts-upgradeable/contracts/utils/PausableUpgradeable.sol#4
                - lib/openzeppelin-contracts/contracts/interfaces/IERC4626.sol#4
                - lib/openzeppelin-contracts/contracts/interfaces/draft-IERC6093.sol#3
                - lib/openzeppelin-contracts/contracts/proxy/utils/Initializable.sol#4
                - lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#4
                - lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4
                - lib/openzeppelin-contracts/contracts/token/ERC20/extensions/ERC4626.sol#4
                - lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#4
                - lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Permit.sol#4
                - lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#4
                - lib/openzeppelin-contracts/contracts/utils/Address.sol#4
                - lib/openzeppelin-contracts/contracts/utils/Context.sol#4
                - lib/openzeppelin-contracts/contracts/utils/ReentrancyGuard.sol#4
                - lib/openzeppelin-contracts/contracts/utils/math/Math.sol#4
        - Version constraint ^0.8.24 is used by:
                - lib/sc_hub-and-spokes-libraries/src/libraries/Errors.sol#2
        - Version constraint 0.8.24 is used by:
                - lib/sc_hub-and-spokes-libraries/src/libraries/TokenHelper.sol#2
                - src/interfaces/IConcreteMultiStrategyVault.sol#2
                - src/interfaces/IStrategy.sol#2
                - src/strategies/StrategyBase.sol#2
        - Version constraint ^0.8.0 is used by:
                - src/interfaces/Errors.sol#2
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
StrategyBase.__StrategyBase_init(IERC20,string,string,address,uint256,address,RewardToken[],address) (src/strategies/StrategyBase.sol#84-132) is never used and should be removed
StrategyBase._getRewardTokens(uint256) (src/strategies/StrategyBase.sol#337-347) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
StrategyBase (src/strategies/StrategyBase.sol#35-403) does not implement functions:
        - StrategyBase._getRewardsToStrategy(bytes) (src/strategies/StrategyBase.sol#402)
        - StrategyBase._handleRewardsOnWithdraw() (src/strategies/StrategyBase.sol#268)
        - StrategyBase._totalAssets() (src/strategies/StrategyBase.sol#401)
        - IStrategy.getAvailableAssetsForWithdrawal() (src/interfaces/IStrategy.sol#12)
        - IStrategy.isProtectStrategy() (src/interfaces/IStrategy.sol#14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Detectors:
The following unused import(s) in lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol should be removed:
        -import {IERC20Permit} from "../extensions/IERC20Permit.sol"; (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#7)
The following unused import(s) in src/strategies/StrategyBase.sol should be removed:
        -import {Initializable} from "@openzeppelin/contracts/proxy/utils/Initializable.sol"; (src/strategies/StrategyBase.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-imports
INFO:Slither:src/strategies/StrategyBase.sol analyzed (27 contracts with 91 detectors), 8 result(s) found
```

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.