// Security Assessment

04.22.2024 - 07.19.2024

Earn V1

Blueprint Finance

HALBORN

Earn V1 - Blueprint Finance

Prepared by: HALBORN

Last Updated: Unknown date

Date of Engagement: April 22nd, 2024 - July 19th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
24	0	0	1	4	19

1. Introduction

Concrete engaged Halborn to conduct a security assessment on their smart contracts. This report includes four assessments carried out on the following schedules:

- Money Printer assessment beginning on April 22nd, 2024 and ending on May 1st, 2024.
- Money Printer Code Updates assessment beginning on May 20th, 2024 and ending on June 5th, 2024.
- Earn diff assessment beginning on June 10th, 2024 and ending on June 17th, 2024.

TABLE OF CONTENTS

- Silo and Radiant Strategies assessment beginning on July 16th, 2024 and ending on July 19th, 2024.

2. Assessment Summary

The team at Halborn was provided one week for the engagement and assigned a full-time security engineer to verify the security of the smart contract. 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 security risks that were mostly addressed/acknowledged by the `Concrete team`.

3. Test Approach And Methodology

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

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Testnet deployment (Foundry).

3.1 Out-Of-Scope

- External libraries and financial-related attacks.
- Files located under **src/examples/*** folder.

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
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 (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C:C)	0 0.25 0.5 0.75 1
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

REPOSITORY

(a) Repository: sc_earn-v1

(b) Assessed Commit ID: 949c177

(c) Items in scope:

- src/vault/ConcreteMultiStrategyVault.sol
- src/swapper/Swapper.sol
- src/swapper/OraclePlug.sol
- src/strategies/StrategyBase.sol
- src/registries/VaultRegistry.sol
- src/registries/TokenRegistry.sol
- src/registries/ImplementationRegistry.sol
- src/queue/WithdrawalQueue.sol
- src/managers/VaultManager.sol
- src/managers/RewardManager.sol
- src/managers/DeploymentManager.sol
- src/factories/VaultFactory.sol
- src/claimRouter/ClaimRouter.sol
- src/strategies/ProtectStrategy.sol
- src/strategies/Aave/AaveV3Strategy.sol
- src/strategies/Aave/DataTypes.sol
- src/strategies/Aave/IAaveV3.sol
- src/strategies/Radiant/RadiantV2Strategy.sol
- src/strategies/Silo/SiloV1Strategy.sol

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

6. ASSESSMENT SUMMARY &

FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM
0	0	1

LOW	INFORMATIONAL
4	19

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LOGIC ISSUE ON VAULT REMOVAL	MEDIUM	SOLVED - 07/29/2024
POTENTIAL DENIAL OF SERVICE ON PREPARE WITHDRAWAL FUNCTION	LOW	RISK ACCEPTED
DENIAL OF SERVICE ON _GETSTRATEGY FUNCTION	LOW	SOLVED - 06/09/2024
CLAIMROUTER CONTRACT IS USING A MOCK INTERFACE	LOW	RISK ACCEPTED
UNCHECKED RETURN VALUES	LOW	SOLVED -

OF ERC20 FUNCTIONS		07/29/2024
SINGLE STEP OWNERSHIP TRANSFER PROCESS	INFORMATIONAL	ACKNOWLEDGED
OWNER CAN RENOUNCE OWNERSHIP	INFORMATIONAL	ACKNOWLEDGED
A SINGLE COMPROMISED ACCOUNT CAN LOCK UP THE PROTOCOL	INFORMATIONAL	ACKNOWLEDGED
POTENTIAL DENIAL OF SERVICE ON REMOVEVAULT WITH BLOCK GAS LIMIT	INFORMATIONAL	SOLVED - 06/09/2024
CHECKED INCREMENTS IN FOR LOOPS INCREASES GAS CONSUMPTION	INFORMATIONAL	SOLVED - 05/07/2024
INCOMPLETE ERROR HANDLING IN RETIRESTRATEGY	INFORMATIONAL	SOLVED - 07/29/2024
MISSING CHECKS FOR INDEX OUT OF BOUNDS	INFORMATIONAL	SOLVED - 05/07/2024

USE EXTERNAL INSTEAD OF PUBLIC METHODS	INFORMATIONAL	SOLVED - 05/07/2024
PUSHO IS NOT SUPPORTED BY ALL CHAINS	INFORMATIONAL	ACKNOWLEDGED
MISSING ZERO ADDRESS CHECKS	INFORMATIONAL	ACKNOWLEDGED
CHECKED INCREMENTS IN FOR LOOPS INCREASES GAS CONSUMPTION	INFORMATIONAL	ACKNOWLEDGED
LACK OF VALIDATION ON PROTECTSTRATEGY	INFORMATIONAL	ACKNOWLEDGED
DUPLICATED IMPORTS	INFORMATIONAL	SOLVED - 07/30/2024
EVENTS FOR FUNCTION CALLS	INFORMATIONAL	ACKNOWLEDGED
CENTRALIZATION RISK: LACK OF ACCESS CONTROL IN REQUESTFUNDS	INFORMATIONAL	ACKNOWLEDGED

MISSING ZERO ADDRESS CHECKS ON CONSTRUCTOR	INFORMATIONAL	SOLVED - 07/29/2024
UNRESTRICTED REWARD ENABLING	INFORMATIONAL	ACKNOWLEDGED
CENTRALIZATION RISK: RETIRESTRATEGY	INFORMATIONAL	ACKNOWLEDGED
NO EVENTS FOR CLAIM FUNCTION	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 LOGIC ISSUE ON VAULT REMOVAL

// MEDIUM

Description

There's a logic issue on `removeVault` on `VaultRegistry.sol`. If the admin of vault manager chooses a wrong address vault and a correct vault ID to be removed (`removeVault(address vault_, bytes32 vaultId_)`), will be removed the wrong ones. This issue could lead to a wrong vault removal due to this logic issue. Consider checking if the vault ID that will be removed corresponds with the vault address passed as first parameter.
Moreover, the value of `vaultIdToAddressArray[vaultId_]` will never be removed from the mapping since the vault is not equal to the value of the mapping on the check inside `_handleRemoveVault` function.

```
68  function removeVault(address vault_, bytes32 vaultId_
69          console.log("VAULT ID TO ADDR ARRAY: ",vaultI
70          vaultExists[vault_] = false;
72          _handleRemoveVault(vault_, vaultIdToAddressAr
73          assert(vaultsByToken[IERC4626(vault_).asset()
74      }
```

Proof of Concept

The following `Foundry` test was used in order to prove the aforementioned issue:

```
function test_removeVaultAndRedeployLogicIssue() public {

        for (uint256 index = 0; index < 10; index++) { //_de
            ImplementationData memory data =
            ImplementationData({implementationAddress: imple


            vm.prank(admin);
            vaultManager.registerNewImplementation(keccak256
```

```
            (bytes memory initData, Strategy[] memory strate
            vm.prank(admin);
            vaultAddress.push(vaultManager.deployNewVault(ke
        }
        for (uint256 index = 0; index < 10; index++) {
            address[] memory vaults = vaultRegistry.getAllVa
            console.log("VAULT: ",vaults[index]);
        }
        //(address vault,) = _deployVault(false);

        vm.prank(admin);
        vaultManager.removeVault(vaultAddress[9], keccak256(

        for (uint256 index = 0; index < 9; index++) {
            address[] memory vaults = vaultRegistry.getAllVa
            console.log("VAULT: ",vaults[index]);
        }

        address[] memory vaults = vaultRegistry.getAllVaults
        console.log("LEN OLD VAULT: ",vaults.length);
        //assertEq(vaults.length, 0, "Length");
        vm.warp(block.timestamp + 1);

        ImplementationData memory data =
        ImplementationData({implementationAddress: implement
        (bytes memory initData, Strategy[] memory strategies
        vm.prank(admin);
        address vaultAddress2 = vaultManager.deployNewVault(

        address[] memory newVaults = vaultRegistry.getAllVau
        console.log("LEN NEW VAULT: ",newVaults.length);

        for (uint256 index = 0; index < 10; index++) {
            address[] memory vaults = vaultRegistry.getAllVa
            console.log("VAULT: ",vaults[index]);
        }

        vm.prank(admin);
        vaultManager.removeVault(vaultAddress2, keccak256(ab
        for (uint256 index = 0; index < 9; index++) {
            address[] memory vaults = vaultRegistry.getAllVa
            console.log("VAULT: ",vaults[index]);
        }
        //assertEq(newVaults.length, 1, "Length");

    }
```

Evidence:

BVSS

AO:S/AC:L/AX:L/C:L/I:C/A:C/D:C/Y:C/R:N/S:C (4.5)

Recommendation

Consider checking that the address exists in both arrays, to make sure that the id and the address are correct.

Remediation Comment

SOLVED : The **Concrete team** solved the issue. The issue is already fixed in the commit id sent. We are checking if the vault address exists for the vaultId. The fix is at the end of the function `_handleRemoveVault`.

7.2 POTENTIAL DENIAL OF SERVICE ON PREPARE WITHDRAWAL FUNCTION

// LOW

Description

There's a logic issue on the `WithdrawaQueue` contract. The

problem arises when the vault wanted to finalize a request withdrawal ID before preparing a withdrawal with a `request ID-1`.

```
147 /// @dev preapares a request to be transferred
148     ///  Emits WithdrawalClaimed event
149     //TODO test this function
150     //-next-line naming-convention
151     function prepareWithdrawal(uint256 _requestId, ui
152         external
153         onlyOwner
154         returns (address recipient, uint256 amount, u
155     {
156         console.log("LAST FINALIZED REQUEST ID: ",las
157         console.log("_requestId: ",_requestId);
158         if (_requestId == 0) revert InvalidRequestId(
160
161         WithdrawalRequest storage request = _requests
162
163         if (request.claimed) revert RequestAlreadyCla
164
165         recipient = request.recipient;
166
167         WithdrawalRequest storage prevRequest = _requ
168
169         amount = request.cumulativeAmount - prevReque
170         console.log("amount: prepare withdrawal: ",am
171         console.log("_avaliableAssets: ",_avaliableAs
172
173         if (_avaliableAssets > amount) {
174             assert(_requestsByOwner[recipient].remove
175             avaliableAssets = _avaliableAssets - amou
176             request.claimed = true;
177             //This is commented to fit the requiremer
178             //instead of this we will call _withdrawS
179             //IERC20(TOKEN).safeTransfer(recipient, r
180
181             emit WithdrawalClaimed(_requestId, recipi
182         }
183     }
```

Since the owner is the vault and the only way the vault can call `_finalize` is through `batchClaimWithdrawal`, the issue has been downgraded from high to low due to the protocol design. However, be sure before production or future releases that the `_finalize` external function is not called by the vault for security reasons.
The `batchClaimWithdrawal` function, is calling first the `prepareWithdrawal` (`claimWithdrawal` private function) instead of `_finalize`.

```
/**
     * @notice Claims multiple withdrawal requests starting
     * @dev This function allows the contract owner to claim
     * @param maxRequests The maximum number of withdrawal r
```

```
     */
    function batchClaimWithdrawal(uint256 maxRequests) exter
        if (address(withdrawalQueue) == address(0)) revert C
        uint256 availableAssets = getAvailableAssetsForWitho

        uint256 lastFinalizedId = withdrawalQueue.getLastFir
        uint256 lastCreatedId = withdrawalQueue.getLastReque
        uint256 newLastFinalized = lastFinalizedId;

        uint256 max = lastCreatedId < lastFinalizedId + maxR
        console.log("MAX: ",max);

        for (uint256 i = lastFinalizedId + 1; i <= max;) {
            uint256 newAvailiableAssets = claimWithdrawal(i,
            // -next-line incorrect-equality
            if (newAvailiableAssets == availableAssets) brea

            availableAssets = newAvailiableAssets;
            newLastFinalized = i;
            unchecked {
                i++;
            }
        }

        if (newLastFinalized != lastFinalizedId) {
            withdrawalQueue._finalize(newLastFinalized);
        }
    }
```

BVSS

AO:S/AC:L/AX:L/C:N/I:C/A:C/D:N/Y:C/R:N/S:C (3.8)

Recommendation

Consider ensuring as internal the `_finalize` function.

Remediation Comment

RISK ACCEPTED: The **Concrete team** accepted the risk of this finding. They will not make the function internal. The vault doesn't have any way to directly call that function. The only way to do it is through impersonation using Foundry or Hardhat.

7.3 DENIAL OF SERVICE ON _GETSTRATEGY FUNCTION

// LOW

Description

There is a potential denial of service (DoS) vulnerability in the `_getStrategy` function. The `vaults` array, which stores all vaults created for a specific token, lacks an upper limit on its size. Consequently, the contract owner could theoretically create an infinite number of vaults. This array retrieves all vaults associated with a particular token, which means an infinite number of vaults could be created for a given token. This would cause a denial of service when the `_getStrategy` function is called due to the excessive size of the `vaults` array.

```
function _getStrategy(address tokenAddress, uint256 amount)
        //retrieves all vaults created for an specific toker

        address[] memory vaults = vaultRegistry.getVaultsByT
        //TODO change to a criteria where when can decidede

        address selectedProtectionStrat = address(0x0);
        bool requiresFunds = true;
        //Iterates over array of vaults to find the best vau
        uint256 len = vaults.length;
        uint256 maxYieldFound = type(uint256).max;
        for (uint256 i; i < len; i++) {
            IConcreteMultiStrategyVault currentVault = ICond
            address protectionStrat = currentVault.protectSt
            //only considers vaults with protection strategi
            if (protectionStrat == address(0x0)) {
                continue;
            }

            uint256 protectionStratYield = IMockStrategy(pro

            //if the protection strategy has enough assets t
            if (
                IMockStrategy(protectionStrat).getAvailableA
                    && (protectionStratYield < maxYieldFound
            ) {
                console.log("hola puto");
                selectedProtectionStrat = protectionStrat;
                requiresFunds = false;
                maxYieldFound = protectionStratYield;
            }
            if (requiresFunds == false) {
                continue;
            }


            if (currentVault.getAvailableAssetsForWithdrawal
                selectedProtectionStrat = protectionStrat;
                requiresFunds = true;
                maxYieldFound = protectionStratYield;
```

```
            }
        }
        return (selectedProtectionStrat, requiresFunds);
    }
```

Gas Consumption Concern

Additionally, the `getVaultsByToken` function consumes a significant amount of gas when handling a large number of vaults. This issue is particularly evident if an unusually large number of vaults are created for a specific token (as demonstrated in the proof of concept).

```
function getVaultsByToken(address asset) external view virtu
        return vaultsByToken[asset].values();
    }
```

Irreversibility of the Operation

The operation is irreversible for the affected token. Once a vast number of vaults are created for a token, all subsequent operations on vaults associated with that token will fail. Given that the operation requires privileged access, the severity of this issue has been downgraded to low.

BVSS

AO:S/AC:L/AX:L/C:N/I:C/A:C/D:N/Y:C/R:N/S:C (3.8)

Recommendation

1. Implement a limit on the number of vaults that can be created for each token.
2. Optimize the `getVaultsByToken` function to handle large arrays more efficiently.
3. Consider adding mechanisms to reverse or mitigate the creation of excessive vaults.

By addressing these recommendations, the potential for a denial of service can be significantly reduced, ensuring the robustness and reliability of the contract.

Remediation Comment

SOLVED: The **Concrete team** solved this issue.

7.4 CLAIMROUTER CONTRACT IS USING A MOCK INTERFACE

// LOW

Description

It has been observed that the user blueprint did not receive the asset tokens, since the `executeBorrowClaim` function is executed without code.

```
/// @notice Function to request assets from the vault.
    /// @dev Requests assets from the vault and executes a b
    /// @param tokenAddress The address of the token.
    /// @param amount_ The amount of assets to request.
    /// @param userBlueprint The address of the user's bluep
    function requestToken(VaultFlags, address tokenAddress,
        external
        onlyBlueprint
    {
        uint256 amount = amount_;
        (address protectionStrat, bool requiresFunds) = _get
        console.log(protectionStrat);
        if (protectionStrat == address(0x0)) {
            //iterates over array
            uint256 len = tokenCascade.length;
            for (uint256 i; i < len; i++) {
                //We avoid using the same token as the one t
                if (tokenAddress == tokenCascade[i]) continu

                //TODO change amount

                amount = _convertFromTokenToStable(tokenAddr

                //We control both the length of the array ar
                // slither-disable-next-line calls-loop
                (protectionStrat, requiresFunds) = _getStrat
                if (protectionStrat != address(0x0)) {
                    break;
                }
            }
        }

        if (protectionStrat == address(0x0)) {
            revert NoProtectionStrategiesFound();
        }
        emit ClaimRequested(protectionStrat, amount, IMockPr
        IMockProtectStrategy(protectionStrat).executeBorrowC
    }
```

MockERC44656Protect.sol:

```
function executeBorrowClaim(uint256 amount, address recipier
        console.log("Entered to empty function from on scope
```

```
        //empty
    }
```

Since it's a mock, the issue has been downgraded, however it's not recommended using a mock contract within a contract. If for somehow this is not updated, then the wrong function will be executed with unexpected errors or behaviors for the users using the money printer.

BVSS

AO:S/AC:L/AX:L/C:N/I:C/A:N/D:N/Y:C/R:N/S:C (3.1)

Recommendation

Consider changing the proper interface `IProtectStrategy` instead of `IMockProtectStrategy` on the scope contract `ClaimRouter` due to could lead to unexpected behavior since the critical code inside `executeBorrowClaim` is never run.

Remediation Comment

RISK ACCEPTED: The **Concrete team** accepted the risk of this finding.

7.5 UNCHECKED RETURN VALUES OF ERC20 FUNCTIONS

// LOW

Description

`ERC20(token).approve` reverted if the underlying ERC20 token approve did not return boolean. When transferring the token, the protocol uses `safeTransfer` and `safeTransferFrom`, but when approving the payout token, the `safeApprove` is not used for non-standard token such as USDT, calling approve will revert because the OpenZeppeling

ERC20 enforces the underlying token return a boolean.

```
function approve(address spender, uint256 value) public virt
        address owner = _msgSender();
        _approve(owner, spender, value);
        return true;
    }
```

While the token such as USDT does not return boolean:

https://etherscan.io/address/0xdac17f958d2ee523a2206206994597c13d831ec7#code#L126

USDT or other ERC20 token that does not return boolean for approve is not supported as the payout token. It has been observed on the `SiloV1Strategy` contract, missing require on the `approve` function.

BVSS

AO:A/AC:L/AX:M/C:N/I:L/A:M/D:M/Y:N/R:P/S:U (2.3)

Recommendation

Consider using `safeApprove` instead of `approve`.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by applying recommendations.

7.6 SINGLE STEP OWNERSHIP TRANSFER PROCESS

// INFORMATIONAL

Description

Ownership of the contracts can be lost as the `RewardManager,TokenRegistry,ImplementationRegistry,DeploymentManager and VaultFactory` contract is inherited from the `Ownable`

contract, and their ownership can be transferred in a single-step process. The address the ownership is changed to should be verified to be active or willing to act as the owner.

```
function transferOwnership(address newOwner) public virtual
    require(newOwner != address(0), "Ownable: new owner is t
    _transferOwnership(newOwner);
}

function _transferOwnership(address newOwner) internal virtu
    address oldOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (1.0)

Recommendation

Consider using the `Ownable2Step` **https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol** library over the `Ownable` library.

Remediation Comment

ACKNOWLEDGED : The **Concrete team** acknowledged the issue. Since each of the contracts is owned by other contracts, the ability to transfer or renounce ownership is significantly limited. The use of `Ownable2Step` requires that the new owner accept ownership. Without a significant re-write of the existing contracts (which would negate the functionality of Ownable2Step) this is not feasible.

7.7 OWNER CAN RENOUNCE OWNERSHIP

// INFORMATIONAL

Description

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities.
The Openzeppelin's Ownable used in this project contract implements renounceOwnership. This can represent a certain risk if the ownership is renounced for any other reason than by design. Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner.

```
function renounceOwnership() public virtual onlyOwner {
    _transferOwnership(address(0));
```

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (1.0)

Recommendation

To mitigate the risks associated with unintended or unauthorized ownership transfers, it is recommended to adopt a more secure ownership transfer mechanism, such as OpenZeppelin's `Ownable2Step`. This implementation requires a two-step process for ownership transfer: first, the current owner initiates the transfer by specifying a new owner, and then the new owner must accept the ownership. This process adds an additional layer of security by ensuring that ownership is only transferred after explicit confirmation from both parties involved.
Implementing `Ownable2Step` will not only enhance the security of the contracts by preventing accidental or malicious ownership transfers but also ensure that any systems or contracts relying on its authorization remain consistent and up-to-date. This recommendation aligns with best practices for contract ownership management and governance within the Ethereum ecosystem.

Remediation Comment

ACKNOWLEDGED : The **Concrete team** acknowledged the issue. Since each of the contracts is owned by other contracts, the ability to transfer or renounce ownership is significantly limited. The use of `Ownable2Step` requires that the new owner accept ownership. Without a significant re-write of the existing contracts (which would negate the functionality of Ownable2Step) this is not feasible.

7.8 A SINGLE COMPROMISED ACCOUNT CAN LOCK UP THE PROTOCOL

// INFORMATIONAL

Description

The ability to `pauseVault()` and `unpauseVault()` the protocol is unilaterally controlled by a single `admin` account, which if compromised could be exploited to attempt to lock up the protocol and extort users.
It is advised that these capabilities are separated into distinct roles in an effort to reduce the consolidation of power that could be wielded in the event of a protocol exploit.

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (1.0)

Recommendation

Consider adopting both an `UNPAUSE_ROLE` and `PAUSE_ROLE`.

Remediation Comment

ACKNOWLEDGED : The **Concrete team** acknowledged the issue. The client belief that the likelihood of this is exceptionally low. The admin role will be granted to a multi-sig to prevent rogue action. If they were to introduce a pause and unpause role, the same multisig would be provided in both roles, negating any positive benefits and creating unnecessary overhead.

7.9 POTENTIAL DENIAL OF SERVICE ON REMOVEVAULT

WITH BLOCK GAS LIMIT

// INFORMATIONAL

Description

It has been observed that if a particular vault wanted to be removed, the `_handleRemoveVault` function iterates over all `vaultArray` consuming a lot of gas in case there's a huge amount of vault created on the array `vaultArray_`. If the array is big enough, the block gas limit will be reached and the transaction will never get processed. For that reason, it is recommended to add a require statement for limiting adding, for example, more than *n* different `vault`.

```
79  function _handleRemoveVault(address vault_, address[
80          uint256 length = vaultArray_.length;
81
82              if (vaultArray_[i] == vault_) {
83                  if (i < length - 1) {
84                      vaultArray_[i] = vaultArray_[leng
85                  }
86                  vaultArray_.pop();
87                  break;
88              }
89              unchecked {
90                  i++;
91              }
92          }
93      }
```

BVSS

AO:S/AC:L/AX:H/C:N/I:N/A:C/D:N/Y:C/R:N/S:C (1.0)

Recommendation

Consider ensuring a fixed created vaults in order to avoid potential denial of service on `_handleRemoveVault` function.

Remediation Comment

SOLVED: The **Concrete team** solved this issue.

7.10 CHECKED INCREMENTS IN FOR LOOPS INCREASES GAS CONSUMPTION

// INFORMATIONAL

Description

Most of the solidity for loops use an **uint256** variable counter that increments by 1 and starts at 0. These increments don't need to be checked for over/underflow because the variable will never reach the max capacity of **uint256** as it would run out of gas long before that happens.

BVSS

AO:A/AC:H/AX:H/C:M/I:N/A:N/D:N/Y:N/R:N/S:U (0.5)

Recommendation

It is recommended to uncheck the increments in for loops to save gas. For example, instead of:

```
538                 Strategy memory strategy = strategies
539                 //We control both the length of the d
540                 //-next-line calls-loop
541                 uint256 withdrawable = strategy.strat
542                 if (diff.mulDiv(strategy.allocation.d
543                     revert InsufficientFunds(strategy
544                 }
545                 uint256 amountToWithdraw = amount_.mu
546                 //We control both the length of the d
547                 //-next-line unused-return,calls-loop
548                 strategy.strategy.withdraw(amountToWi
549                 totalWithdrawn += amountToWithdraw;
550             }
```

Use:

```
69  for (uint256 i; i < len;) {
70                  Strategy memory strategy = strategies

72                  //-next-line calls-loop
73                  uint256 withdrawable = strategy.strat
74                  if (diff.mulDiv(strategy.allocation.d
```

```
75                      revert InsufficientFunds(strategy
76                  }
77                  uint256 amountToWithdraw = amount_.mu
78                  //We control both the length of the d
79                  //-next-line unused-return,calls-loop
80                  strategy.strategy.withdraw(amountToWi
81                  totalWithdrawn += amountToWithdraw;
82                  unchecked {++i};
83              }
```

Remediation Comment

SOLVED : The **Concrete team** solved the issue by adding `unchecked` statement.

7.11 INCOMPLETE ERROR HANDLING IN RETIRESTRATEGY

// INFORMATIONAL

Description

The function `retireStrategy` calls `_protocolWithdraw(balanceOfUnderlying(shares), 0)` without verifying that `balanceOfUnderlying` returns a valid value. This could potentially cause issues if `balanceOfUnderlying` does not return the expected value.

```
149 function retireStrategy() external onlyOwner {
150     _getRewardsToStrategy();
151     uint256 shares = collateralToken.balanceOf(addres
153 }
```

BVSS

AO:A/AC:H/AX:H/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (0.3)

Recommendation

Consider adding proper checks on the return of `balanceOfUnderlying` function.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by applying recommendations.

7.12 MISSING CHECKS FOR INDEX OUT OF BOUNDS

// INFORMATIONAL

Description

The functions `pullFundsFromSingleStrategy`, `pushFundsIntoSingleStrategy`, and `addStrategy` fail to implement checks for out-of-bounds access on the array of strategies with which the vault can interact.

```
/**
     * @notice Pulls funds back from a single strategy into
     * @dev Can only be called by the vault owner.
     * @param index_ The index of the strategy from which to
     */
    function pullFundsFromSingleStrategy(uint256 index_) ext
        Strategy memory strategy = strategies[index_];
        // slither-disable-next-line unused-return
        strategy.strategy.redeem(strategy.strategy.balanceOf
        if (!IERC20(asset()).approve(address(strategy.strate
    }

    /**
     * @notice Pushes funds from the vault into a single str
     * @dev Can only be called by the vault owner. Reverts i
     * @param index_ The index of the strategy into which to
     */
    function pushFundsIntoSingleStrategy(uint256 index_) ext
        if (vaultIdle) revert VaultIsIdle();
        Strategy memory strategy = strategies[index_];
        // slither-disable-next-line unused-return
        strategies[index_].strategy.deposit(
            totalAssets().mulDiv(strategy.allocation.amount,
        );
    }
```

When the `replace_` parameter is set to true, the function did not perform a check on the index.

```
function addStrategy(uint256 index_, bool replace_, Strategy
        external
        nonReentrant
        onlyOwner
        takeFees
    {
        //Ensure that allotments do not total > 100%
        uint256 allotmentTotals = 0;
        uint256 len = strategies.length;
        for (uint256 i; i < len; i++) {
            allotmentTotals += strategies[i].allocation.amou
        }

        if (replace_) {
            if (allotmentTotals - strategies[index_].allocat
                revert AllotmentTotalTooHigh();
            }
            // slither-disable-next-line unused-return
            strategies[index_].strategy.redeem(
                strategies[index_].strategy.balanceOf(addres
            );
            if (!IERC20(asset()).approve(address(strategies[
```

```
            emit StrategyReplaced(strategies[index_], newStr

            strategies[index_] = newStrategy_;
            if (!IERC20(asset()).approve(address(newStrategy
        } else {
            if (allotmentTotals + newStrategy_.allocation.am
                revert AllotmentTotalTooHigh();
            }
            strategies.push(newStrategy_);
            if (!IERC20(asset()).approve(address(newStrategy
        }
        emit StrategyAdded(newStrategy_);
    }
```

BVSS

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 advisable to implement out-of-bounds checks when accessing arrays in Solidity. Additionally, providing an error message when attempting to fetch data beyond the array's limit can inform the user about the cause of the execution reversion.

Remediation Comment

SOLVED : The **Concrete team** solved the issue by adding array index checks.

7.13 USE EXTERNAL INSTEAD OF PUBLIC METHODS

// INFORMATIONAL

Description

The following methods are public and could be external. External is more gas optimize that public, so it should be used as much as possible.

- swapTokensForReward() should be declared external:
  - Swapper.swapTokensForReward() (swapper/Swapper.sol#75-94).
- setRewardManager() should be declared external:
  - Swapper.setRewardManager() (swapper/Swapper.sol#102-105).
- disableTokenForSwap() should be declared external:
  - Swapper.disableTokenForSwap() (swapper/Swapper.sol#111-113).

BVSS

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

Recommendation

Use external instead of public methods.

Remediation Comment

SOLVED : The **Concrete team** solved the issue by replacing public to external.

7.14 PUSH0 IS NOT SUPPORTED BY ALL CHAINS

// INFORMATIONAL

Description

The compiler for Solidity 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

BVSS

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

Recommendation

It is important to consider the targeted deployment chains before writing smart contracts because, in the future, there might exist a need for deploying the contracts in a network that could not support new opcodes from Shanghai or Cancun EVM versions.

Remediation Comment

ACKNOWLEDGED : The **Concrete team** acknowledged the issue. It is their stance that this a low probability over a given time horizon. Additionally, if the client elects to deploy to a chain that is incompatible with Shanghai, it can be created a contract that utilizes an older Solidity version.

7.15 MISSING ZERO ADDRESS

CHECKS

// INFORMATIONAL

Description

Some missing zero addresses have been found:

1. The `addVault` function allows the addition of a vault with a zero address.
2. The strategy struct permits passing a zero address during its initialization.

```
43  function addVault(address vault_, bytes32 vaultId_) e
44          if (vaultExists[vault_]) {
45              revert VaultAlreadyExists();
46          }
47          vaultExists[vault_] = true;
49          allVaultsCreated.push(vault_);
50          assert(vaultsByToken[IERC4626(vault_).asset()
51          console.log("VAULT ADDED: ",vault_);
52          emit VaultAdded(vault_, vaultId_);
53      }
```

```
188 function initialize(
189         IERC20 baseAsset_,
190         string memory shareName_,
191         string memory shareSymbol_,
192         Strategy[] memory strategies_,
193         address feeRecipient_,
194         VaultFees memory fees_,
195         uint256 depositLimit_,
196         address owner_
197     ) external initializer nonReentrant {
198         __ERC4626_init(IERC20Metadata(address(baseAss
199         __ERC20_init(shareName_, shareSymbol_);
200         __Ownable_init(owner_);
201         _queue_init();
202         if (address(baseAsset_) == address(0)) revert
203
204         // Loop through provided strategies, validate
205         uint256 len = strategies_.length;
206         //console.log("Strategies len: ",len);
207         for (uint256 i; i < len;) {
208             //We control both the length of the array
209             // slither-disable-next-line calls-loop
210             if (strategies_[i].strategy.asset() != ad
211                 revert VaultAssetMismatch();
212             }
213             //console.log("Protection strats on concr
214             if (strategies_[i].strategy.isProtectStra
215                 if (protectStrategy != address(0x0))
216                 protectStrategy = address(strategies_
217             }
219             //We control both the length of the array
```

```
220                 // slither-disable-next-line calls-loop
221                 if (!baseAsset_.approve(address(strategie
222                     revert ERC20ApproveFail();
223                 }
224                 unchecked {
225                     i++;
226                 }
227             }
```

BVSS

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

Recommendation

To address the identified issues of allowing zero addresses in the `addVault` and `initialize` functions, you need to add checks to ensure that zero addresses are not passed for `vault_` in `addVault` and for `strategy` and other relevant addresses in `initialize`.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding.

7.16 CHECKED INCREMENTS IN FOR LOOPS INCREASES GAS CONSUMPTION

// INFORMATIONAL

Description

Most of the solidity for loops use an **uint256** variable counter that increments by 1 and starts at 0. These increments don't need to be checked for over/underflow because the variable will never reach the max capacity of **uint256** as it would run out of gas long before that happens.

BVSS

AV:R/AC:L/AX:L/PR:N/UI:N/C:N/I:N/A:N/D:N/Y:N/R:N/S:U

(0.0)

Recommendation

It is recommended to uncheck the increments in for loops to save gas. For example, instead of:

```
280             address protectionStrat = IConcreteMultiS
281
282             //only considers vaults with protection s
283             if (protectionStrat == address(0x0)) {
284                 continue;
285             }
286
287             lastProtectionStrat = protectionStrat;
288             uint256 stratBorrowDebt = IMockProtectStr
289
290             if (stratBorrowDebt == 0) continue;
291
292             uint256 amountToBeSent = amount_.mulDiv(s
293
294             //this function is only called by the blu
295             totalSent += amountToBeSent;
296             //TODO HAndle cases where the amount sent
297             //the funtion should be able to set the r
298
299             //slither-disable-next-line arbitrary-ser
300             IERC20(tokenAddress).safeTransferFrom(use
301             if (isReward) {
302                 emit RewardAdded(protectionStrat, amo
303                 continue;
304             }
305
306             if (amountToBeSent > stratBorrowDebt) {
307                 emit RewardAdded(protectionStrat, amo
308                 emit Repayment(protectionStrat, strat
309                 IMockProtectStrategy(protectionStrat)
310             } else {
311                 emit Repayment(protectionStrat, amour
312                 IMockProtectStrategy(protectionStrat)
313             }
314         }
```

Use:

```
279 for (uint256 i; i < vaults.length; ) {
280             address protectionStrat = IConcreteMultiS
281
282             //only considers vaults with protection s
283             if (protectionStrat == address(0x0)) {
284                 continue;
285             }
286
```

```
287             lastProtectionStrat = protectionStrat;
288             uint256 stratBorrowDebt = IMockProtectStr
289
290             if (stratBorrowDebt == 0) continue;
291
292             uint256 amountToBeSent = amount_.mulDiv(s
293
294             //this function is only called by the blu
295             totalSent += amountToBeSent;
296             //TODO HAndle cases where the amount sent
297             //the funtion should be able to set the r
298
299             //slither-disable-next-line arbitrary-ser
300             IERC20(tokenAddress).safeTransferFrom(use
301             if (isReward) {
302                 emit RewardAdded(protectionStrat, amo
303                 continue;
304             }
305
306             if (amountToBeSent > stratBorrowDebt) {
307                 emit RewardAdded(protectionStrat, amo
308                 emit Repayment(protectionStrat, strat
309                 IMockProtectStrategy(protectionStrat)
310             } else {
311                 emit Repayment(protectionStrat, amour
312                 IMockProtectStrategy(protectionStrat)

314          unchecked {++i};
315         }
```

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding.

7.17 LACK OF VALIDATION ON PROTECTSTRATEGY

// INFORMATIONAL

Description

It has been observed that no parameter passed into the constructor was properly validated.

```
67          IERC20 baseAsset_,
68          address feeRecipient_,
69          address owner_,
70          uint256 rewardFee_,
71          address claimRouter_,
```

```
72          address vault_
73      ) {
74          IERC20Metadata metaERC20 = IERC20Metadata(add
75
76          rewardFee = rewardFee_;
77          //This can be initially set to zero and then
78          //slither-disable-next-line missing-zero-chec
79          claimRouter = claimRouter_;
80
81          RewardToken[] memory rewardTokenEmptyArray =
82          __StrategyBase_init(
83              baseAsset_,
84              string.concat("Concrete Earn Protect ", n
85              string.concat("ctPct-", metaERC20.symbol(
86              feeRecipient_,
87              type(uint256).max,
88              owner_,
89              rewardTokenEmptyArray,
90              vault_
91          );
92          //slither-disable-next-line unused-return
93          //baseAsset_.approve(claimRouter, type(uint25
94      }
```

The only parameter that can be set again is `claimRouter_` through the `setClaimRouter` function by the owner.

BVSS

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

Recommendation

Consider implementing proper validation for all parameters passed into the constructor. This will enhance the security and robustness of the contract by ensuring that invalid or malicious input cannot compromise its functionality. Specifically, consider adding checks to verify the validity of addresses, ensuring they are not zero addresses, and confirming that the `rewardFee_` is within an acceptable range.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding.

7.18 DUPLICATED IMPORTS

// INFORMATIONAL

Description

The `SafeERC20` and `Math` libraries are imported twice on the `StrategyBase` contract. This should be cleaned up to avoid confusion.

```
import {
    ERC4626Upgradeable,
    ERC20Upgradeable as ERC20,
    IERC20,
    IERC20Metadata,
    IERC4626
} from "@openzeppelin/contracts-upgradeable/token/ERC20/exte
import {Initializable} from "@openzeppelin/contracts/proxy/u
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20
import {Math} from "@openzeppelin/contracts/utils/math/Math.
import {ReentrancyGuard} from "@openzeppelin/contracts/utils
import {PausableUpgradeable} from "@openzeppelin/contracts-u
import {OwnableUpgradeable} from "@openzeppelin/contracts-up
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20
import {Context} from "@openzeppelin/contracts/utils/Context
import {ContextUpgradeable} from "@openzeppelin/contracts-up
import {Math} from "@openzeppelin/contracts/utils/math/Math.
import {Errors} from "../interfaces/Errors.sol";
import {ReturnedRewards} from "../interfaces/IStrategy.sol";
import {
    VaultFees,
    VaultInitParams,
    Strategy,
    IConcreteMultiStrategyVault
} from "../interfaces/IConcreteMultiStrategyVault.sol";
```

BVSS

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

Recommendation

Consider removing the duplicated imports.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by applying recommendations.

7.19 EVENTS FOR FUNCTION CALLS

// INFORMATIONAL

Description

Consider to add events for critical state-changing functions to improve traceability and debugging.

```
/**
     * @notice Sets the recipient address for protocol fees.
     * @dev Can only be called by the contract owner.
     * @param feeRecipient_ The address to which protocol fe
     */
    function setFeeRecipient(address feeRecipient_) external
        if (feeRecipient_ == address(0)) revert InvalidFeeRe
        feeRecipient = feeRecipient_;
    }

    /**
     * @notice Sets the maximum limit for deposits into the
     * @dev Can only be called by the contract owner.
     * @param depositLimit_ The maximum amount that can be d
     */
    //TODO: Add events for these functions
    //-next-line events-maths
    function setDepositLimit(uint256 depositLimit_) external
        if (depositLimit_ == 0) revert InvalidDepositLimit()
        depositLimit = depositLimit_;
    }
```

BVSS

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

Recommendation

Consider adding events:

```
event FeeRecipientUpdated(address indexed newFeeRecipient);
event DepositLimitUpdated(uint256 newDepositLimit);

function setFeeRecipient(address feeRecipient_) external onl
    if (feeRecipient_ == address(0)) revert InvalidFeeRecipi
    feeRecipient = feeRecipient_;
    emit FeeRecipientUpdated(feeRecipient_);
}

function setDepositLimit(uint256 depositLimit_) external onl
    if (depositLimit_ == 0) revert InvalidDepositLimit();
```

```
    depositLimit = depositLimit_;
    emit DepositLimitUpdated(depositLimit_);
}
```

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding.

7.20 CENTRALIZATION RISK: LACK OF ACCESS CONTROL IN REQUESTFUNDS

// INFORMATIONAL

Description

The `requestFunds` function is marked with `onlyProtect`, but there's no mechanism to change or update the `protectStrategy` address once it is set. This could be a problem if the `protectStrategy` needs to be updated or changed due to unforeseen issues or upgrades.

```
function requestFunds(uint256 amount) external onlyProtect {
        //tries toi send from balance
        uint256 availableAssets = IERC20(asset()).balanceOf(
        uint256 acumulated = availableAssets;
        if (availableAssets < amount) {
            acumulated = availableAssets;
            uint256 len = strategies.length;
            for (uint256 i; i < len; i++) {
                IStrategy currentStrategy = strategies[i].st

                if (address(strategies[i].strategy) == prote
                    continue;
                }

                uint256 pending = amount - acumulated;

                //calculate the max we can get from the stra
                uint256 avaliableInStrat = currentStrategy.g

                if (avaliableInStrat == 0) {
                    continue;
                }

                uint256 toWithdraw = pending;
                if (avaliableInStrat < pending) {
```

```
                    toWithdraw = avaliableInStrat;
                }

                acumulated += toWithdraw;

                //We control both the length of the array ar
                //-next-line unused-return,calls-loop
                currentStrategy.withdraw(toWithdraw, address
                if (acumulated >= amount) {
                    break;
                }
            }
        }

        //after requesting funds deposits them into the prot
        if (acumulated < amount) {
            revert InsufficientFunds(IStrategy(address(this)
        }
        //-next-line unused-return
        IStrategy(protectStrategy).deposit(amount, address(t
        emit RequestedFunds(protectStrategy, amount);
    }
```

BVSS

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

Recommendation

Consider addressing the limitation of not being able to update the `protectStrategy` address, it is recommended to introduce a function that allows for the modification of the `protectStrategy` address. This function should be restricted to a privileged role, such as the contract owner or an admin, to ensure security and prevent unauthorized changes.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding.

7.21 MISSING ZERO ADDRESS CHECKS ON CONSTRUCTOR

// INFORMATIONAL

Description

It has been observed missing zero address checks on the `Radiant` and `Silo` constructors:

```
26  constructor(
27          IERC20 baseAsset_,
28          address feeRecipient_,
29          address owner_,
30          uint256 rewardFee_,
31          address siloAsset_,
32          address siloRepository_,
33          address siloIncentivesController_,
34          address[] memory extraRewardAssets,
35          uint256[] memory extraRewardFees,
36          address vault_
37      ) {
38          IERC20Metadata metaERC20 = IERC20Metadata(add
39
40
41          silo = ISilo(siloRepository.getSilo(siloAsset
42          if (address(silo) == address(0)) revert Inval
43          // validate the bridge asset
44          if (siloAsset_ != address(baseAsset_)) {
45              address[] memory siloAssets_ = silo.getAs
46              uint256 length = siloAssets_.length;
47              uint256 i;
48              while (i < length) {
49                  if (siloAssets_[i] == address(baseAss
50                      break;
51                  }
52                  unchecked {
53                      i++;
54                  }
55              }
56              if (i == length) revert AssetDivergence()
57          }
58          siloIncentivesController = ISiloIncentivesCon
59          ISilo.AssetStorage memory assetStorage = silo
60          collateralToken = IERC20(assetStorage.collate
61
62          // prepare rewardTokens array
63
64          uint256 rewardsLength = extraRewardAssets.len
65          RewardToken[] memory rewardTokenArray = new R
66          // assign the silo reward token first and the
67          address[] memory rewards = getRewardTokenAddr
68          rewardTokenArray[0] = RewardToken(IERC20(rewa
69
70          for (uint256 i = 0; i < rewardsLength;) {
71              rewardTokenArray[i+1] = RewardToken(IERC2
72              unchecked {
73                  i++;
74              }
75          }
76
77          if (address(collateralToken) == address(0)) r
78          __StrategyBase_init(
79              baseAsset_,
80              string.concat("Concrete Earn SiloV1 ", me
81              string.concat("ctSlV1-", metaERC20.symbol
82              feeRecipient_,
83              type(uint256).max,
```

```
            owner_,
            rewardTokenArray,
            vault_
        );
        //slither-disable-next-line unused-return
        baseAsset_.approve(address(silo), type(uint25
    }
```

```
constructor(
        IERC20 baseAsset_,
        address feeRecipient_,
        address owner_,
        uint256 rewardFee_,
        address addressesProvider_,
        address vault_
    ) {
        IERC20Metadata metaERC20 = IERC20Metadata(add

        addressesProvider = ILendingPoolAddressesProv
        lendingPool = ILendingPool(addressesProvider.
                                                        
        rToken = IAToken(reserveData.aTokenAddress);
        if (rToken.UNDERLYING_ASSET_ADDRESS() != addr
            revert AssetDivergence();
        }
        rewardsEnabled = false;
        incentiveController = IChefIncentivesControll

        __StrategyBase_init(
            baseAsset_,
            string.concat("Concrete Earn RadiantV2 ",
            string.concat("ctRdV2-", metaERC20.symbol
            feeRecipient_,
            type(uint256).max,
            owner_,
            _getRewardTokens(rewardFee_),
            vault_
        );
        //slither-disable-next-line unused-return
        baseAsset_.approve(address(lendingPool), type
    }
```

BVSS

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

Recommendation

Consider adding proper checks on the constructor.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by applying recommendations.

7.22 UNRESTRICTED REWARD ENABLING

// INFORMATIONAL

Description

The `setEnableRewards` function allows the owner to enable or disable rewards without any checks. If rewards are enabled when they should not be, this could lead to issues.

```
130 | /**
131 |      * @dev by setting rewardsEnabled to true the str
133 |      * https://docs.radiant.capital/radiant/project-i
134 |      */
135 |      function setEnableRewards(bool _rewardsEnabled)
136 |         rewardsEnabled = _rewardsEnabled;
137 |         emit SetEnableRewards(msg.sender, _rewardsEnd
138 |     }
```

BVSS

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

Recommendation

Consider as the owner verifies eligibility before enabling rewards. Adding a require statement might help to enforce checks:

```
function setEnableRewards(bool _rewardsEnabled) external onl
    // Add any necessary checks here
    require(checkEligibilityCriteria(), "Not eligible for re

    rewardsEnabled = _rewardsEnabled;
    emit SetEnableRewards(msg.sender, _rewardsEnabled);
}

function checkEligibilityCriteria() internal view returns (b
    // Implement the eligibility check logic
}
```

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged the issue.

7.23 CENTRALIZATION RISK: RETIRESTRATEGY

// INFORMATIONAL

Description

Although `retireStrategy` is restricted to the owner, retiring a strategy is a critical function. Additional checks and safeguards should be considered to prevent misuse.
Centralization Risk Analysis

1. **Owner Control:**
   - **Risk:** The owner has significant control over the contract, including the ability to retire the strategy at any time. This centralization means that users of the strategy must trust the owner to act in their best interests.
   - **Mitigation:** Consider implementing a multi-signature wallet for the owner role to ensure that multiple parties must agree before critical functions like `retireStrategy` can be executed.
2. **Sudden Withdrawal:**
   - **Risk:** The `retireStrategy` function can withdraw all assets from the strategy, potentially without prior notice to the users. This could disrupt users' investment plans and cause a sudden loss of yield.
   - **Mitigation:** Implement a time-lock mechanism that delays the execution of the `retireStrategy` function, providing users with notice and an opportunity to withdraw their funds if they choose to.

BVSS

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

Recommendation

To address the centralization risks and enhance the security of the `retireStrategy` function, consider the following improvements:

1. **Time-Lock Mechanism:** Introduce a delay between the initiation and execution of the `retireStrategy` function.

```
uint256 public retireStrategyTimestamp;
uint256 public constant TIMELOCK_DURATION = 2 days;

function initiateRetireStrategy() external onlyOwner {
    retireStrategyTimestamp = block.timestamp + TIMELOCK_DUR
}

function executeRetireStrategy() external onlyOwner {
    require(block.timestamp >= retireStrategyTimestamp, "Tim
    _getRewardsToStrategy();
    _protocolWithdraw(rToken.balanceOf(address(this)), 0);
    retireStrategyTimestamp = 0; // Reset the timestamp
}
```

2. **Emergency Withdraw Mechanism:** Allow users to withdraw their funds in case of emergency, without waiting for the owner to retire the strategy.

```
function emergencyWithdraw() external {
    uint256 userBalance = balanceOf(msg.sender);
    _protocolWithdraw(userBalance, 0);
    _transfer(msg.sender, userBalance);
}
```

3. **Notification System:** Implement a notification system to alert users when the `retireStrategy` function is initiated.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged the issue.

7.24 NO EVENTS FOR CLAIM FUNCTION

// INFORMATIONAL

Description

The `claim` function in `_getRewardsToStrategy` uses a try-catch block, which is a good practice. However, consider logging the error for better traceability.

```
function _getRewardsToStrategy() internal override {
        if (!rewardsEnabled) return;
        if (address(incentiveController) == address(0)) retu
        address[] memory _assets = new address[](1);
        _assets[0] = address(rToken);

        //slither-disable-next-line unused-return
        try incentiveController.claim(address(this), _assets
    }
```

BVSS

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

Recommendation

Consider logging the error for better traceability.

```
try incentiveController.claim(address(this), _assets) {} cat
    emit ClaimFailed(reason);
}

event ClaimFailed(bytes reason);
```

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged the issue.

---

addressing potential vulnerabilities introduced by code modifications.