

Earn V1 *Concrete*

HALBORN

Earn V1 - Concrete

Prepared by:  HALBORN

Last Updated 10/15/2024

Date of Engagement by: 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

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
 - 3.1 Out-of-scope
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
 - 7.1 Logic issue on vault removal
 - 7.2 Potential denial of service on prepare withdrawal function
 - 7.3 Unchecked return values of erc20 functions
 - 7.4 Denial of service on _getstrategy function
 - 7.5 Claimrouter contract is using a mock interface
 - 7.6 Single step ownership transfer process
 - 7.7 Owner can renounce ownership
 - 7.8 A single compromised account can lock up the protocol
 - 7.9 Potential denial of service on removevault with block gas limit
 - 7.10 Checked increments in for loops increases gas consumption

- 7.11 Incomplete error handling in retirestrategy
- 7.12 Missing checks for index out of bounds
- 7.13 Use external instead of public methods
- 7.14 Push0 is not supported by all chains
- 7.15 Missing zero address checks
- 7.16 Checked increments in for loops increases gas consumption
- 7.17 Lack of validation on protectstrategy
- 7.18 Duplicated imports
- 7.19 Events for function calls
- 7.20 Centralization risk: lack of access control in requestfunds
- 7.21 Missing zero address checks on constructor
- 7.22 Unrestricted reward enabling
- 7.23 Centralization risk: retirestrategy
- 7.24 No events for claim function

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.
- 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

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

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND 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 TokenNameRegistry.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:

REMEDIATION COMMIT ID:

^

- ef76d69
- b285ec0
- <https://github.com/Blueprint-Finance/money-printer/pull/25/commits/7e47dfd39dba59e02fd9b236390321c2c5dba0a0>
- <https://github.com/Blueprint-Finance/money-printer/pull/26/commits/167bbc342f1b836763767b0dfbc503bc4240f7ad>

- <https://github.com/Blueprint-Finance/money-printer/pull/27/commits/2306e730a5deeee15eb77f6f2aa3d5e61a0d0ff2>
- 949c177

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	4	19

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LOGIC ISSUE ON VAULT REMOVAL	MEDIUM	SOLVED - 07/30/2024
POTENTIAL DENIAL OF SERVICE ON PREPARE WITHDRAWAL FUNCTION	LOW	RISK ACCEPTED
UNCHECKED RETURN VALUES OF ERC20 FUNCTIONS	LOW	SOLVED - 07/30/2024
DENIAL OF SERVICE ON _GETSTRATEGY FUNCTION	LOW	SOLVED - 06/10/2024
CLAIMROUTER CONTRACT IS USING A MOCK INTERFACE	LOW	RISK ACCEPTED
SINGLE STEP OWNERSHIP TRANSFER PROCESS	INFORMATIONAL	ACKNOWLEDGED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
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 - 06/10/2024
CHECKED INCREMENTS IN FOR LOOPS INCREASES GAS CONSUMPTION	INFORMATIONAL	SOLVED - 05/08/2024
INCOMPLETE ERROR HANDLING IN RETIRESTRATEGY	INFORMATIONAL	SOLVED - 07/30/2024
MISSING CHECKS FOR INDEX OUT OF BOUNDS	INFORMATIONAL	SOLVED - 05/08/2024
USE EXTERNAL INSTEAD OF PUBLIC METHODS	INFORMATIONAL	SOLVED - 05/08/2024
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

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF VALIDATION ON PROTECTSTRATEGY	INFORMATIONAL	ACKNOWLEDGED
DUPLICATED IMPORTS	INFORMATIONAL	SOLVED - 07/31/2024
EVENTS FOR FUNCTION CALLS	INFORMATIONAL	ACKNOWLEDGED
CENTRALIZATION RISK: LACK OF ACCESS CONTROL IN REQUESTFUNDS	INFORMATIONAL	ACKNOWLEDGED
MISSING ZERO ADDRESS CHECKS ON CONSTRUCTOR	INFORMATIONAL	SOLVED - 07/30/2024
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_) external onlyOwner
69 |         console.log("VAULT ID TO ADDR ARRAY: ", vaultIdToAddressArray[vault_]);
70 |         vaultExists[vault_] = false;
71 |         _handleRemoveVault(vault_, allVaultsCreated);
72 |         _handleRemoveVault(vault_, vaultIdToAddressArray[vaultId_]);
73 |         assert(vaultsByToken[IERC4626(vault_).asset()].remove(vault_));
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++) { // _deployVault(false)
        ImplementationData memory data =
            ImplementationData({implementationAddress: implementation, initDataRequired: true});
        vm.prank(admin);
        vaultManager.registerNewImplementation(keccak256(abi.encode("Test", index)), data);

        (bytes memory initData, Strategy[] memory strategies) = _getInitData();
        vm.prank(admin);
        vaultAddress.push(vaultManager.deployNewVault(keccak256(abi.encode("Test",
```

```
index)), initData));
}

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

vm.prank(admin);
vaultManager.removeVault(vaultAddress[9], keccak256(abi.encode("Test", 8)));
//admin wanted to remove the vault ID 8, however the vault removed was vault 9.

for (uint256 index = 0; index < 9; index++) {
    address[] memory vaults = vaultRegistry.getAllVaults();
    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: implementation, initDataRequired:
true});
(bytes memory initData, Strategy[] memory strategies) = _getInitData();
vm.prank(admin);
address vaultAddress2 = vaultManager.deployNewVault(keccak256(abi.encode("Test",
9)), initData);

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

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

vm.prank(admin);
vaultManager.removeVault(vaultAddress2, keccak256(abi.encode("Test", 9)));
for (uint256 index = 0; index < 9; index++) {
    address[] memory vaults = vaultRegistry.getAllVaults();
    console.log("VAULT: ", vaults[index]);
}
```

```
//assertEq(newVaults.length, 1, "Length");
}
```

Evidence:

```
Ran 1 test for test/VaultManager.t.sol:VaultManagerTest
[PASS] test_removeVaultAndRedeployLogicIssue() (gas: 64724956)
Logs:
VAULT: 0xd239Fd8522B9A5B783E813BDB6D724D69A045dF1
VAULT: 0x1d7e3CD690e1D6d10048d244b17AB4CF679bBec0
VAULT: 0x73aA8a7c0970a2014127f1ae2696b160dfb37F54
VAULT: 0x9Fe19dDB5410e6D0836497238BF274FacAc98820
VAULT: 0x66aae17Ae4c469cCA50dcEcb3CE8af51893B30E0
VAULT: 0xE9c739480FB2C313F3f0034cC6D45f42904d502
VAULT: 0x4bC7b0d7Cd3e29f4E4a43608d5fbcbE316Da7F4a
VAULT: 0x80d8D968621a1bC402b5B2b1e4D22a55F9360aa2
VAULT: 0x8eFECF53C18e147acE6b20892Cb6e99cC403e4c2
VAULT: 0x3Cfc31F79A89476A29f68A2B76a4B8572076EA71
VAULT ID TO ADDR ARRAY: 0x8eFECF53C18e147acE6b20892Cb6e99cC403e4c2
VAULT LEN: 10
VAULT LEN: 1
VAULT: 0xd239Fd8522B9A5B783E813BDB6D724D69A045dF1
VAULT: 0x1d7e3CD690e1D6d10048d244b17AB4CF679bBec0
VAULT: 0x73aA8a7c0970a2014127f1ae2696b160dfb37F54
VAULT: 0x9Fe19dDB5410e6D0836497238BF274FacAc98820
VAULT: 0x66aae17Ae4c469cCA50dcEcb3CE8af51893B30E0
VAULT: 0xE9c739480FB2C313F3f0034cC6D45f42904d502
VAULT: 0x4bC7b0d7Cd3e29f4E4a43608d5fbcbE316Da7F4a
VAULT: 0x80d8D968621a1bC402b5B2b1e4D22a55F9360aa2
VAULT: 0x8eFECF53C18e147acE6b20892Cb6e99cC403e4c2
```



1. Admin wanted to remove this vault

2. However, the vault removed was
0x3Cfc31F79A89476A29f68A2B76a4B857
2076EA71

BVSS

A0: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

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 `WithdrawalQueue` 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 prepares a request to be transferred
148     /// Emits WithdrawalClaimed event
149     //TODO test this function
150     //-next-line naming-convention
151     function prepareWithdrawal(uint256 _requestId, uint256 _availableAssets)
152         external
153         onlyOwner
154         returns (address recipient, uint256 amount, uint256 availableAssets)
155     {
156         console.log("LAST FINALIZED REQUEST ID: ",lastFinalizedRequestId)
157         console.log("_requestId: ",_requestId);
158         if (_requestId == 0) revert InvalidRequestId(_requestId);
159         if (_requestId < lastFinalizedRequestId) revert RequestNotFoundOr
160
161         WithdrawalRequest storage request = _requests[_requestId];
162
163         if (request.claimed) revert RequestAlreadyClaimed(_requestId);
164
165         recipient = request.recipient;
166
167         WithdrawalRequest storage prevRequest = _requests[_requestId - 1]
168
169         amount = request.cumulativeAmount - prevRequest.cumulativeAmount;
170         console.log("amount: prepare withdrawal: ",amount);
171         console.log("_availableAssets: ",_availableAssets);
172
173         if (_availableAssets > amount) {
174             assert(_requestsByOwner[recipient].remove(_requestId));
175             availableAssets = _availableAssets - amount;
176             request.claimed = true;
177             //This is commented to fit the requirements of the vault
178             //instead of this we will call _withdrawStrategyFunds
179             //IERC20(TOKEN).safeTransfer(recipient, realAmount);
180 }
```

```
180
181         emit WithdrawalClaimed(_requestId, recipient, amount);
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 from the
 * lastFinalizedRequestId.
 * @dev This function allows the contract owner to claim multiple withdrawal
 * requests in batches.
 * @param maxRequests The maximum number of withdrawal requests to be processed in
 * this batch.
 */
function batchClaimWithdrawal(uint256 maxRequests) external onlyOwner nonReentrant {
    if (address(withdrawalQueue) == address(0)) revert QueueNotSet();
    uint256 availableAssets = getAvailableAssetsForWithdrawal();

    uint256 lastFinalizedId = withdrawalQueue.getLastFinalizedRequestId();
    uint256 lastCreatedId = withdrawalQueue.getLastRequestId();
    uint256 newLastFinalized = lastFinalizedId;

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

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

        availableAssets = newAvailableAssets;
        newLastFinalized = i;
        unchecked {
            i++;
        }
    }
}
```

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

BVSS

A0: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

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 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 OpenZeppelin ERC20 enforces the underlying token return a boolean.

```
function approve(address spender, uint256 value) public virtual returns (bool) {
    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.

89 | `baseAsset_.approve(address(silo), type(uint256).max);`

BVSS

A0:A/AC:L/AX:M/C:N/I:L/A:M/D:M/Y:N/R:P/S:U (3.8)

Recommendation

Consider using `safeApprove` instead of `approve`.

Remediation

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

Remediation Hash

ef76d69792cc6aec1d60982536bab8af21992608

7.4 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) internal returns (address, bool) {
    //retrieves all vaults created for an specific token

    address[] memory vaults = vaultRegistry.getVaultsByToken(tokenAddress);
    //TODO change to a criteria where when can decidede base on yield

    address selectedProtectionStrat = address(0x0);
    bool requiresFunds = true;
    //Iterates over array of vaults to find the best vault to withdraw from
    uint256 len = vaults.length;
    uint256 maxYieldFound = type(uint256).max;
    for (uint256 i; i < len; i++) {
        IConcreteMultiStrategyVault currentVault =
IConcreteMultiStrategyVault(vaults[i]);
        address protectionStrat = currentVault.protectStrategy();
        //only considers vaults with protection strategies
        if (protectionStrat == address(0x0)) {
            continue;
        }

        uint256 protectionStratYield =
IMockStrategy(protectionStrat).highWatermark();

        //if the protection strategy has enough assets to withdraw we are done with
        the search
        if (
            IMockStrategy(protectionStrat).getAvailableAssetsForWithdrawal() >=
amount
            && (protectionStratYield < maxYieldFound || requiresFunds == true)
        )
            selectedProtectionStrat = protectionStrat;
            maxYieldFound = protectionStratYield;
            requiresFunds = false;
    }

    return (selectedProtectionStrat, requiresFunds);
}
```

```

) {
    console.log("hola puto");
    selectedProtectionStrat = protectionStrat;
    requiresFunds = false;
    maxYieldFound = protectionStratYield;
}
if (requiresFunds == false) {
    continue;
}

if (currentVault.getAvailableAssetsForWithdrawal() >= amount &&
protectionStratYield < maxYieldFound) {
    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 virtual returns (address[] memory
vaults) {
    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

A0:S/AC:L/AX:L/C:N/I:C/A:C/D:N/Y:C/R:N/S:C (3.6)

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

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

Remediation Hash

b285ec09faf4ac4c213e59f2f9a1f49fb7414424

7.5 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 borrow claim through the
protection strategy.

/// @param tokenAddress The address of the token.
/// @param amount_ The amount of assets to request.
/// @param userBlueprint The address of the user's blueprint contract.

function requestToken(VaultFlags, address tokenAddress, uint256 amount_, address
payable userBlueprint)
    external
    onlyBlueprint
{
    uint256 amount = amount_;
    (address protectionStrat, bool requiresFunds) = _getStrategy(tokenAddress,
amount);
    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 that failed
            if (tokenAddress == tokenCascade[i]) continue;

            //TODO change amount

            amount = _convertFromTokenToStable(tokenAddress, amount_);

            //We control both the length of the array and the external call
            // slither-disable-next-line calls-loop
            (protectionStrat, requiresFunds) = _getStrategy(tokenCascade[i],
amount);
            if (protectionStrat != address(0x0)) {
                break;
            }
        }
    }
}
```

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

MockERC44656Protect.sol:

```
function executeBorrowClaim(uint256 amount, address recipient) external {
    console.log("Entered to empty function from on scope contract");
    //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

A0: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

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

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 onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _transferOwnership(newOwner);
}

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

BVSS

A0: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

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.

Remediation Hash

<https://github.com/Blueprint-Finance/money-printer/pull/25/commits/7e47dfd39dba59e02fd9b236390321c2c5dba0a0>

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

A0: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

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.

Remediation Hash

<https://github.com/Blueprint-Finance/money-printer/pull/25/commits/7e47dfd39dba59e02fd9b236390321c2c5dba0a0>

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

A0: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

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.

Remediation Hash

<https://github.com/Blueprint-Finance/money-printer/pull/25/commits/7e47dfd39dba59e02fd9b236390321c2c5dba0a0>

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[] storage vaultArray
80 |                               uint256 length = vaultArray_.length;
81 |     for (uint256 i = 0; i < length;) {
82 |         if (vaultArray_[i] == vault_) {
83 |             if (i < length - 1) {
84 |                 vaultArray_[i] = vaultArray_[length - 1];
85 |             }
86 |             vaultArray_.pop();
87 |             break;
88 |         }
89 |         unchecked {
90 |             i++;
91 |         }
92 |     }
93 | }
```

BVSS

A0: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

SOLVED: The Concrete team solved this issue.

Remediation Hash

b285ec09faf4ac4c213e59f2f9a1f49fb7414424

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

A0: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:

```
537 for (uint256 i; i < len; i++) {  
538     Strategy memory strategy = strategies[i];  
539     //We control both the length of the array and the external  
540     //--next-line calls-loop  
541     uint256 withdrawable = strategy.strategy.previewRedeem(st  
542     if (diff.mulDiv(strategy.allocation.amount, 10_000, Math.  
543         revert InsufficientFunds(strategy.strategy, diff * st  
544     }  
545     uint256 amountToWithdraw = amount_.mulDiv(strategy.alloca  
546     //We control both the length of the array and the external  
547     //--next-line unused-return,calls-loop  
548     strategy.strategy.withdraw(amountToWithdraw, receiver_, a  
549     totalWithdrawn += amountToWithdraw;  
550 }
```

Use:

```
69 for (uint256 i; i < len;) {  
70     Strategy memory strategy = strategies[i];  
71     //We control both the length of the array and the external  
72     //--next-line calls-loop  
73     uint256 withdrawable = strategy.strategy.previewRedeem(st  
74     if (diff.mulDiv(strategy.allocation.amount, 10_000, Math.  
75         revert InsufficientFunds(strategy.strategy, diff * st  
76     }  
77 }
```

```
77     uint256 amountToWithdraw = amount_.mulDiv(strategy.alloca
78     //We control both the length of the array and the externa
79     //‐next‐line unused‐return,calls‐loop
80     strategy.strategy.withdraw(amountToWithdraw, receiver_, a
81     totalWithdrawn += amountToWithdraw;
82     unchecked {++i};
83 }
```

Remediation

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

Remediation Hash

<https://github.com/Blueprint-Finance/money-printer/pull/25/commits/7e47dfd39dba59e02fd9b236390321c2c5dba0a0>

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(address(this));  
152 |         _protocolWithdraw(balanceOfUnderlying(shares), 0);  
153 |     }
```

BVSS

A0: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

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

Remediation Hash

ef76d69792cc6aec1d60982536bab8af21992608

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 the vault.  
 * @dev Can only be called by the vault owner.  
 * @param index_ The index of the strategy from which to pull funds.  
 */  
function pullFundsFromSingleStrategy(uint256 index_) external onlyOwner {  
    Strategy memory strategy = strategies[index_];  
    // slither-disable-next-line unused-return  
    strategy.strategy.redeem(strategy.strategy.balanceOf(address(this)),  
address(this), address(this));  
    if (!IERC20(asset()).approve(address(strategy.strategy), 0)) revert  
ERC20ApproveFail();  
}  
  
/**  
 * @notice Pushes funds from the vault into a single strategy based on its  
allocation.  
 * @dev Can only be called by the vault owner. Reverts if the vault is idle.  
 * @param index_ The index of the strategy into which to push funds.  
 */  
function pushFundsIntoSingleStrategy(uint256 index_) external onlyOwner {  
    if (vaultIdle) revert VaultIsIdle();  
    Strategy memory strategy = strategies[index_];  
    // slither-disable-next-line unused-return  
    strategies[index_].strategy.deposit(  
        totalAssets().mulDiv(strategy.allocation.amount, 10_000,  
Math.Rounding.Floor), address(this)  
    );  
}
```

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 calldata newStrategy_)  
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.amount;
    }

    if (replace_) {
        if (allotmentTotals - strategies[index_].allocation.amount + newStrategy_.allocation.amount > 10000) {
            revert AllotmentTotalTooHigh();
        }
        // slither-disable-next-line unused-return
        strategies[index_].strategy.redeem(
            strategies[index_].strategy.balanceOf(address(this)), address(this),
            address(this)
        );
        if (!IERC20(asset()).approve(address(strategies[index_].strategy), 0))
revert ERC20ApproveFail();

        emit StrategyReplaced(strategies[index_], newStrategy_);

        strategies[index_] = newStrategy_;
        if (!IERC20(asset()).approve(address(newStrategy_.strategy),
type(uint256).max)) revert ERC20ApproveFail();
    } else {
        if (allotmentTotals + newStrategy_.allocation.amount > 10000) {
            revert AllotmentTotalTooHigh();
        }
        strategies.push(newStrategy_);
        if (!IERC20(asset()).approve(address(newStrategy_.strategy),
type(uint256).max)) revert ERC20ApproveFail();
    }
    emit StrategyAdded(newStrategy_);
}

```

Score

A0: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

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

Remediation Hash

<https://github.com/Blueprint-Finance/money-printer/pull/26/commits/167bbc342f1b836763767b0dfbc503bc4240f7ad>

7.13 USE EXTERNAL INSTEAD OF PUBLIC METHODS

// INFORMATIONAL

Description

The following methods are public and could be external. External is more gas optimize than 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).

Score

A0: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

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

Remediation Hash

<https://github.com/Blueprint-Finance/money-printer/pull/27/commits/2306e730a5deeee15eb77f6f2aa3d5e61a0d0ff2>

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.

Score

A0: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

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.

Remediation Hash

<https://github.com/Blueprint-Finance/money-printer/pull/25/commits/7e47dfd39dba59e02fd9b236390321c2c5dba0a0>

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_) external override onl
44 |         if (vaultExists[vault_]) {
45 |             revert VaultAlreadyExists();
46 |         }
47 |         vaultExists[vault_] = true;
48 |         vaultIdToAddressArray[vaultId_].push(vault_);
49 |         allVaultsCreated.push(vault_);
50 |         assert(vaultsByToken[IERC4626(vault_).asset()].add(vault_));
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(baseAsset_)));
199 |         __ERC20_init(shareName_, shareSymbol_);
200 |         __Ownable_init(owner_);
201 |         _queue_init();
202 |         if (address(baseAsset_) == address(0)) revert InvalidAssetAddress
203 |
204 |         // Loop through provided strategies, validate them, and approve t
205 |         uint256 len = strategies_.length;
206 |         //console.log("Strategies len: ",len);
207 |         for (uint256 i; i < len;) {
208 |             }
```

```
209     //We control both the length of the array and the external ca
210     // slither-disable-next-line calls-loop
211     if (strategies_[i].strategy.asset() != address(baseAsset_)) {
212         revert VaultAssetMismatch();
213     }
214     //console.log("Protection strats on concret init: ",strategie
215     if (strategies_[i].strategy.isProtectStrategy()) {
216         if (protectStrategy != address(0x0)) revert MultipleProte
217         protectStrategy = address(strategies_[i].strategy);
218     }
219     strategies.push(strategies_[i]);
220     //We control both the length of the array and the external ca
221     // slither-disable-next-line calls-loop
222     if (!baseAsset_.approve(address(strategies_[i].strategy), typ
223         revert ERC20ApproveFail();
224     }
225     unchecked {
226         i++;
227     }
}
```

Score

A0: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

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.

Score

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:

```
279 for (uint256 i; i < vaults.length; i++) {  
280     address protectionStrat = IConcreteMultiStrategyVault(vaults[  
281  
282         //only considers vaults with protection strategies  
283         if (protectionStrat == address(0x0)) {  
284             continue;  
285         }  
286  
287         lastProtectionStrat = protectionStrat;  
288         uint256 stratBorrowDebt = IMockProtectStrategy(protectionStrat)  
289  
290         if (stratBorrowDebt == 0) continue;  
291  
292         uint256 amountToBeSent = amount_.mulDiv(stratBorrowDebt, tota  
293  
294         //this function is only called by the blueprint so we can tru  
295         totalSent += amountToBeSent;  
296         //TODO HAndle cases where the amount sent is more the one exp  
297         //the funtion should be able to set the rest as rewards in th  
298  
299         //slither-disable-next-line arbitrary-send-erc20  
300         IERC20(tokenAddress).safeTransferFrom(userBlueprint, protecti  
301         if (isReward) {  
302             emit RewardAdded(protectionStrat, amountToBeSent);  
303             continue;  
304         }  
305     }  
306 }
```

```

304
305
306     if (amountToBeSent > stratBorrowDebt) {
307         emit RewardAdded(protectionStrat, amountToBeSent - stratB
308         emit Repayment(protectionStrat, stratBorrowDebt);
309         IMockProtectStrategy(protectionStrat).updateBorrowDebt(st
310     } else {
311         emit Repayment(protectionStrat, amountToBeSent);
312         IMockProtectStrategy(protectionStrat).updateBorrowDebt(am
313     }
314 }

```

Use:

```

279 for (uint256 i; i < vaults.length; ) {
280     address protectionStrat = IConcreteMultiStrategyVault(vaults[
281
282     //only considers vaults with protection strategies
283     if (protectionStrat == address(0x0)) {
284         continue;
285     }
286
287     lastProtectionStrat = protectionStrat;
288     uint256 stratBorrowDebt = IMockProtectStrategy(protectionStr
289
290     if (stratBorrowDebt == 0) continue;
291
292     uint256 amountToBeSent = amount_.mulDiv(stratBorrowDebt, tota
293
294     //this function is only called by the blueprint so we can tru
295     totalSent += amountToBeSent;
296     //TODO HAndle cases where the amount sent is more the one exp
297     //the funtion should be able to set the rest as rewards in th
298
299     //slither-disable-next-line arbitrary-send-erc20
300     IERC20(tokenAddress).safeTransferFrom(userBlueprint, protecti
301     if (isReward) {
302         emit RewardAdded(protectionStrat, amountToBeSent);
303         continue;
304     }
305
306     if (amountToBeSent > stratBorrowDebt) {
307         emit RewardAdded(protectionStrat, amountToBeSent - stratB
308         emit Repayment(protectionStrat, stratBorrowDebt);

```

```
309     IMockProtectStrategy(protectiveStrat).updateBorrowDebt(st
310     } else {
311         emit Repayment(protectiveStrat, amountToBeSent);
312         IMockProtectStrategy(protectiveStrat).updateBorrowDebt(am
313     }
314     unchecked {++i};
315 }
```

Remediation

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.

```
66  constructor(
67      IERC20 baseAsset_,
68      address feeRecipient_,
69      address owner_,
70      uint256 rewardFee_,
71      address claimRouter_,
72      address vault_
73  ) {
74      IERC20Metadata metaERC20 = IERC20Metadata(address(baseAsset_));
75
76      rewardFee = rewardFee_;
77      //This can be initially set to zero and then updated by the owner
78      //slither-disable-next-line missing-zero-check
79      claimRouter = claimRouter_;
80
81      RewardToken[] memory rewardTokenEmptyArray = new RewardToken[](0)
82      __StrategyBase_init(
83          baseAsset_,
84          string.concat("Concrete Earn Protect ", metaERC20.symbol()),
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(uint256).max);
94  }
```

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

Score

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

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/extensions/ERC4626Upgradeable.sol";
import {Initializable} from "@openzeppelin/contracts/proxy/utils/Initializable.sol";
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import {Math} from "@openzeppelin/contracts/utils/math/Math.sol";
import {ReentrancyGuard} from "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import {PausableUpgradeable} from "@openzeppelin/contracts-
upgradeable/utils/PausableUpgradeable.sol";
import {OwnableUpgradeable} from "@openzeppelin/contracts-
upgradeable/access/OwnableUpgradeable.sol";
import {SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import {Context} from "@openzeppelin/contracts/utils/Context.sol";
import {ContextUpgradeable} from "@openzeppelin/contracts-
upgradeable/utils/ContextUpgradeable.sol";
import {Math} from "@openzeppelin/contracts/utils/math/Math.sol";
import {Errors} from "../interfaces/Errors.sol";
import {ReturnedRewards} from "../interfaces/IStrategy.sol";
import {
    VaultFees,
    VaultInitParams,
    Strategy,
    IConcreteMultiStrategyVault
} from "../interfaces/IConcreteMultiStrategyVault.sol";
```

Score

A0: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

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

Remediation Hash

949c1773ba1c9ae3ec4813f4444134445ef537d7

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 fees will be sent.  
 */  
function setFeeRecipient(address feeRecipient_) external onlyOwner {  
    if (feeRecipient_ == address(0)) revert InvalidFeeRecipient();  
    feeRecipient = feeRecipient_;  
}  
  
/**  
 * @notice Sets the maximum limit for deposits into the strategy.  
 * @dev Can only be called by the contract owner.  
 * @param depositLimit_ The maximum amount that can be deposited.  
 */  
//TODO: Add events for these functions  
//--next-line events--maths  
function setDepositLimit(uint256 depositLimit_) external onlyOwner {  
    if (depositLimit_ == 0) revert InvalidDepositLimit();  
    depositLimit = depositLimit_;  
}
```

Score

A0: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 onlyOwner {  
    if (feeRecipient_ == address(0)) revert InvalidFeeRecipient();  
    feeRecipient = feeRecipient_;
```

```
emit FeeRecipientUpdated(feeRecipient_);  
}  
  
function setDepositLimit(uint256 depositLimit_) external onlyOwner {  
    if (depositLimit_ == 0) revert InvalidDepositLimit();  
    depositLimit = depositLimit_;  
    emit DepositLimitUpdated(depositLimit_);  
}
```

Remediation

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 to send from balance
    uint256 availableAssets = IERC20(asset()).balanceOf(address(this));
    uint256 accumulated = availableAssets;
    if (availableAssets < amount) {
        accumulated = availableAssets;
        uint256 len = strategies.length;
        for (uint256 i; i < len; i++) {
            IStrategy currentStrategy = strategies[i].strategy;

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

            uint256 pending = amount - accumulated;

            //calculate the max we can get from the strategy
            uint256 availableInStrat =
currentStrategy.getAvailableAssetsForWithdrawal();

            if (availableInStrat == 0) {
                continue;
            }

            uint256 toWithdraw = pending;
            if (availableInStrat < pending) {
                toWithdraw = availableInStrat;
            }

            accumulated += toWithdraw;

        }
    }
}
```

```
//-next-line unused-return,calls-loop
currentStrategy.withdraw(toWithdraw, address(this), address(this));
if (acumulated >= amount) {
    break;
}
}

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

Score

A0: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

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(address(baseAsset_));
39 |
40 |     siloRepository = ISiloRepository(siloRepository_);
41 |     silo = ISilo(siloRepository.getSilo(siloAsset_));
42 |     if (address(silo) == address(0)) revert InvalidAssetAddress();
43 |     // validate the bridge asset
44 |     if (siloAsset_ != address(baseAsset_)) {
45 |         address[] memory siloAssets_ = silo.getAssets();
46 |         uint256 length = siloAssets_.length;
47 |         uint256 i;
48 |         while (i < length) {
49 |             if (siloAssets_[i] == address(baseAsset_)) {
50 |                 break;
51 |             }
52 |             unchecked {
53 |                 i++;
54 |             }
55 |         }
56 |         if (i == length) revert AssetDivergence();
57 |     }
58 |     siloIncentivesController = ISiloIncentivesController(siloIncentiv
59 |     ISilo.AssetStorage memory assetStorage = silo.assetStorage(address
60 |     collateralToken = IERC20(assetStorage.collateralToken);
61 | }
```

```
61
62     // prepare rewardTokens array
63
64     uint256 rewardsLength = extraRewardAssets.length;
65     RewardToken[] memory rewardTokenArray = new RewardToken[](rewardsLength);
66     // assign the silo reward token first and then process the extra
67     address[] memory rewards = getRewardTokenAddresses();
68     rewardTokenArray[0] = RewardToken(IERC20(rewards[0]), rewardFee_,
69
70     for (uint256 i = 0; i < rewardsLength;) {
71         rewardTokenArray[i+1] = RewardToken(IERC20(extraRewardAssets[unchecked {
72             i++;
73         }
74     }
75
76     if (address(collateralToken) == address(0)) revert InvalidAssetAddress;
77     __StrategyBase_init(
78         baseAsset_,
79         string.concat("Concrete Earn SiloV1 ", metaERC20.symbol()),
80         string.concat("ctSlv1-", metaERC20.symbol()),
81         feeRecipient_,
82         type(uint256).max,
83         owner_,
84         rewardTokenArray,
85         vault_
86     );
87     //slither-disable-next-line unused-return
88     baseAsset_.approve(address(silo), type(uint256).max);
89
90 }
```

```
28     constructor(
29         IERC20 baseAsset_,
30         address feeRecipient_,
31         address owner_,
32         uint256 rewardFee_,
33         address addressesProvider_,
34         address vault_
35     ) {
36         IERC20Metadata metaERC20 = IERC20Metadata(address(baseAsset_));
37
38         addressesProvider = ILendingPoolAddressesProvider(addressesProvider);
39         lendingPool = ILendingPool(addressesProvider.getLendingPool());
```

```
40     DataTypes.ReserveData memory reserveData = lendingPool.getReserveData();
41     rToken = IAToken(reserveData.aTokenAddress);
42     if (rToken.UNDERLYING_ASSET_ADDRESS() != address(baseAsset_)) {
43         revert AssetDivergence();
44     }
45     rewardsEnabled = false;
46     incentiveController = IChefIncentivesController(rToken.getIncentiveController());
47
48     __StrategyBase_init(
49         baseAsset_,
50         string.concat("Concrete Earn RadiantV2 ", metaERC20.symbol()),
51         string.concat("ctRdV2-", metaERC20.symbol()),
52         feeRecipient_,
53         type(uint256).max,
54         owner_,
55         _getRewardTokens(rewardFee_),
56         vault_
57     );
58     //slither-disable-next-line unused-return
59     baseAsset_.approve(address(lendingPool), type(uint256).max);
60 }
```

Score

A0: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

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

Remediation Hash

ef76d69792cc6aec1d60982536bab8af21992608

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 strategy will be able to
132   * check the eligibility criteria before enabling rewards here:
133   * https://docs.radiant.capital/radiant/project-info/dlp/eligibility
134   */
135  function setEnableRewards(bool _rewardsEnabled) external onlyOwner {
136      rewardsEnabled = _rewardsEnabled;
137      emit SetEnableRewards(msg.sender, _rewardsEnabled);
138  }
```

Score

A0: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 onlyOwner {
    // Add any necessary checks here
    require(checkEligibilityCriteria(), "Not eligible for rewards");

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

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

Remediation

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.

Score

A0: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_DURATION;
}

function executeRetireStrategy() external onlyOwner {
    require(block.timestamp >= retireStrategyTimestamp, "Timelock not expired");
    _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

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)) return;
    address[] memory _assets = new address[](1);
    _assets[0] = address(rToken);

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

Score

A0: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) {} catch (bytes memory reason) {
    emit ClaimFailed(reason);
}

event ClaimFailed(bytes reason);
```

Remediation

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

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.