// Security Assessment

07.16.2024 - 07.19.2024

Blueprint: Silo and Radiant Strategies

Blueprint Finance

HALBORN

Blueprint: Silo and Radiant Strategies - Blueprint Finance

Prepared by: HALBORN

Last Updated Unknown date

Date of Engagement: July 16th, 2024 - July 19th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
6	0	0	0	1	5

TABLE OF CONTENTS

1. Introduction

`Concrete` engaged Halborn to conduct a security assessment on their smart contract beginning on July 16th, 2024 and ending on July 19th, 2024. A security assessment on smart contracts was performed on the scoped smart contracts provided to the Halborn team.

2. Assessment Summary

The team at Halborn was provided 4 days 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 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.

4. RISK METHODOLOGY

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

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

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

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

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

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

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

ATTACK COMPLEXITY (AX):

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

METRICS:

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

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

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

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

INTEGRITY (I):

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

AVAILABILITY (A):

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

DEPOSIT (D):

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

YIELD (Y):

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

METRICS:

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

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

Impact I is calculated using the following formula:

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

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

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

SCOPE (S):

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

METRICS:

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

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

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

The score is rounded up to 1 decimal places.

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

5. SCOPE

REPOSITORY

(a) Repository: money-printer

(b) Assessed Commit ID: 949c177

(c) Items in scope:

- 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	LOW
0	0	0	1

INFORMATIONAL
5

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNCHECKED RETURN VALUES OF ERC20 FUNCTIONS	LOW	SOLVED - 07/29/2024
INCOMPLETE ERROR HANDLING IN RETIRESTRATEGY	INFORMATIONAL	SOLVED - 07/29/2024
MISSING ZERO ADDRESS CHECKS ON CONSTRUCTOR	INFORMATIONAL	SOLVED - 07/29/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNRESTRICTED REWARD ENABLING	INFORMATIONAL	ACKNOWLEDGED
CENTRALIZATION RISK: RETIRESTRATEGY	INFORMATIONAL	ACKNOWLEDGED
NO EVENTS FOR CLAIM FUNCTION	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 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 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

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 Plan

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

7.2 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

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 Plan

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

7.3 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(siloIncentivesController_);
59          ISilo.AssetStorage memory assetStorage = silo.assetStorage(address(baseAsset_));
60          collateralToken = IERC20(assetStorage.collateralToken);
61
62          // prepare rewardTokens array
63
64          uint256 rewardsLength = extraRewardAssets.length;
65          RewardToken[] memory rewardTokenArray = new RewardToken[](rewardsLength + 1);
66          // assign the silo reward token first and then process the extra reward tokens
67          address[] memory rewards = getRewardTokenAddresses();
68          rewardTokenArray[0] = RewardToken(IERC20(rewards[0]), rewardFee_, 0);
69
70          for (uint256 i = 0; i < rewardsLength;) {
71              rewardTokenArray[i+1] = RewardToken(IERC20(extraRewardAssets[i]), extraRewardFees[
72              unchecked {
73                  i++;
74              }
75          }
76
77          if (address(collateralToken) == address(0)) revert InvalidAssetAddress();
78          __StrategyBase_init(
79              baseAsset_,
80              string.concat("Concrete Earn SiloV1 ", metaERC20.symbol(), " Strategy"),
81              string.concat("ctSlV1-", metaERC20.symbol()),
82              feeRecipient_,
83              type(uint256).max,
84              owner_,
85              rewardTokenArray,
86              vault_
87          );
88          //slither-disable-next-line unused-return
89
```

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

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

        addressesProvider = ILendingPoolAddressesProvider(addressesProvider_);
        lendingPool = ILendingPool(addressesProvider.getLendingPool());
        DataTypes.ReserveData memory reserveData = lendingPool.getReserveData(address(baseAsse
        rToken = IAToken(reserveData.aTokenAddress);
        if (rToken.UNDERLYING_ASSET_ADDRESS() != address(baseAsset_)) {
            revert AssetDivergence();
        }
        rewardsEnabled = false;
        incentiveController = IChefIncentivesController(rToken.getIncentivesController());

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

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 Plan

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

7.4 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 handle rdnt rewards
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     }
```

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 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 Plan

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

7.5 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_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 Plan

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

7.6 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 {}
    }
```

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

event ClaimFailed(bytes reason);
```

Remediation Plan

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

8. AUTOMATED TESTING

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base. The slither issues are considered false positives and well-known by the client added on the source code using `slither-disable-next-line` comment.

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.