// Security Assessment

09.17.2025 - 09.29.2025

Upgradable Multisig and Queue Changes

Blueprint Finance

HALBORN

Upgradable Multisig and Queue Changes - Blueprint Finance

Prepared by: HALBORN

Last Updated 10/06/2025

Date of Engagement: September 17th, 2025 - September 29th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
10	0	0	1	3	6

TABLE OF CONTENTS

1. Introduction

`Blueprint Finance` engaged `Halborn` to perform a security assessment of their smart contracts from September 17th, 2025 to September 29th, 2025. The assessment scope was limited to the smart contracts provided to Halborn. Commit hashes and additional details are available in the Scope section of this report.

The `Blueprint Finance` codebase in scope consists of smart contracts implementing a multi-strategy ERC4626 compliant vault with modular fee logic, strategy allocation, configurable whitelist controls, and a multi-signature custody strategy.

2. Assessment Summary

`Halborn` was allocated 9 days for this engagement and assigned `1` full-time security engineer to conduct a comprehensive review of the smart contracts within scope. The engineer is an expert in blockchain and smart contract security, with advanced skills in penetration testing and smart contract exploitation, as well as extensive knowledge of multiple blockchain protocols.

The objectives of this assessment are to:

- Identify potential security vulnerabilities within the smart contracts.
- Verify that the smart contract functionality operates as intended.

In summary, `Halborn` identified several areas for improvement to reduce the likelihood and impact of security risks, which were mostly acknowledged by the `Blueprint Finance team`. The primary recommendations were as follows:

- `Update the reward calculation to exclude shares pending withdrawal from totalSupply() when distributing rewards, ensuring only active shares are considered.`
- `Align the rounding logic in previewWithdraw with the actual withdrawal calculation by using Math.Rounding.Ceil for the fee.`
- `Implement a mechanism to prevent immediate withdrawals after deposits, such as a short lockup period or withdrawal delay, to ensure users cannot game the reward distribution.`
- `Enforce the rate change threshold in unpauseAndAdjustTotalAssets() by calling _validateRateChange() before updating asset balances.`

3. Test Approach And Methodology

`Halborn` conducted a combination of manual code review and automated security testing to balance efficiency, timeliness, practicality, and accuracy within the scope of this assessment. While manual testing is crucial for identifying flaws in logic, processes, and implementation, automated testing enhances coverage of smart contracts and quickly detects deviations from established security best practices.

The following phases and associated tools were employed throughout the term of the assessment:

- Research into the platform's architecture, purpose and use.
- Manual code review and walkthrough of smart contracts to identify any logical issues.
- Comprehensive assessment of the safety and usage of critical Solidity variables and functions within scope that could lead to arithmetic-related vulnerabilities.
- Local testing using custom scripts (`Foundry`).
- Fork testing against main networks (`Foundry`).
- Static security analysis of scoped contracts, and imported functions (`Slither`).

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: sc_earn-v1

(b) Assessed Commit ID: 51248b8

(c) Items in scope:

- src/strategies/MultiSigStrat/MultiSigStrategy.sol
- src/strategies/StrategyBase.sol
- src/vault/ConcreteMultiStrategyVault.sol

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

REMEDIATION COMMIT ID:

- 509af24

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	1	3

INFORMATIONAL
6

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
BURNED SHARES CAUSE REWARD LOSS DURING PENDING WITHDRAWALS	MEDIUM	NOT APPLICABLE - 10/01/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PREVIEWWITHDRAW UNDERESTIMATES REQUIRED SHARES DUE TO ROUNDING MISMATCH	LOW	SOLVED - 10/01/2025
REWARD ALLOCATION SUSCEPTIBLE TO SNIPING	LOW	NOT APPLICABLE - 10/01/2025
MISSING RATE CHANGE VALIDATION IN ASSET ADJUSTMENT AFTER UNPAUSE	LOW	RISK ACCEPTED - 10/01/2025
WITHDRAWAL QUEUE MAXREDEEM DOES NOT ACCOUNT FOR WITHDRAWAL FEE	INFORMATIONAL	ACKNOWLEDGED - 10/01/2025
CENTRALIZATION RISKS	INFORMATIONAL	ACKNOWLEDGED - 10/01/2025
MISSING VALIDATION FOR REWARD TOKEN AND VAULT FEE VALUES	INFORMATIONAL	ACKNOWLEDGED - 10/01/2025
NON-STANDARD/FEE-ON-TRANSFER TOKENS CAN INFLATE SHARES	INFORMATIONAL	ACKNOWLEDGED - 10/01/2025
FEE BYPASS WHEN FEERECIPIENT IS THE CALLER	INFORMATIONAL	ACKNOWLEDGED - 10/01/2025
RECEIVER NOT VALIDATED AGAINST WHITELIST	INFORMATIONAL	ACKNOWLEDGED - 10/01/2025

7. FINDINGS & TECH DETAILS

7.1 BURNED SHARES CAUSE REWARD LOSS DURING PENDING WITHDRAWALS

// MEDIUM

Description

In the `ConcreteMultiStrategyVault` when `isQueueMandatory` is enabled, withdrawals are processed through a queue. When a user requests a withdrawal, their shares are burned immediately in the `_redeem()` function:

```
666 function _redeem(uint256 sharesToRedeem, address receiver_, address owner_, uint256 feeShares, u
667     private
668 {
669     if (withdrawalsPaused) revert WithdrawalsPaused();
670     if (msg.sender != owner_) {
671         _spendAllowance(owner_, msg.sender, sharesToRedeem);
672     }
673     _burn(owner_, sharesToRedeem);
674     if (feeShares > 0) _mint(feeRecipient, feeShares);
675     uint256 availableAssetsForWithdrawal = getAvailableAssetsForWithdrawal();
676     WithdrawalQueueHelper.processWithdrawal(
677         assets,
678         sharesToRedeem - feeShares,
679         receiver_,
680         availableAssetsForWithdrawal,
681         asset(),
682         address(withdrawalQueue),
683         minQueueRequest,
684         strategies,
685         parkingLot,
686         isQueueMandatory
687     );
688     emit Withdraw(msg.sender, receiver_, owner_, assets, sharesToRedeem);
689 }
```

However, the vault's `totalSupply()` function still includes these pending withdrawal shares until the owner/operator processes the queue with `batchClaimWithdrawal()`:

```
731 function totalSupply() public view override(ERC20Upgradeable, IERC20) returns (uint256 total) {
732     total = VaultActionsHelper.getTotalSupply(super.totalSupply(), withdrawalQueue);
733 }
```

If `harvestRewards()` is called while there are pending withdrawals, the reward calculation uses a `totalSupply_` that includes shares which have already been burned and are no longer eligible for rewards.

```
1134 function harvestRewards(bytes calldata encodedData) external nonReentrant onlyOwner {
1135   RewardsHelper.harvestRewards(strategies, encodedData, totalSupply(), rewardIndex, rewardAddres
1136   emit RewardsHarvested();
1137 }
```

This causes a portion of the rewards to become unclaimable in the protocol. Remaining users also receive less than their fair share.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:M (5.0)

Recommendation

Update the reward calculation to exclude shares pending withdrawal from `totalSupply()` when distributing rewards, ensuring only active shares are considered.

Remediation Comment

NOT APPLICABLE: It has been concluded that this issue is not applicable, as stated by the **Blueprint Finance team**:

> *We use the queue only for multisig strategies, and usually the on-chain strategies (Aave/Compound) are used for atomic vault purposes. Second, we are not using on-chain rewards; instead, we focus on off-chain rewards calculation based on user shares. So far, all our vaults rely on off-chain rewards. Also, v2 vaults are coming, and while the rewards logic is part of the initial architecture, it is currently not in use. Since it is not a pressing issue, we would like to skip it. We do not have rewards on the multisig strategy; therefore, this issue is not applicable.*

7.2 PREVIEWWITHDRAW UNDERESTIMATES REQUIRED SHARES DUE TO ROUNDING MISMATCH

// LOW

Description

The `previewWithdraw()` function in `ConcreteMultiStrategyVault` is intended to let users estimate how many shares they need to burn to withdraw a given amount of assets. However, it uses `Math.Rounding.Floor` when calculating the withdrawal fee:

```
776 function previewWithdraw(uint256 assets_) public view override returns (uint256 shares) {
777     shares = _convertToShares(assets_, Math.Rounding.Ceil);
778     shares = msg.sender != feeRecipient
779         ? shares.mulDiv(MAX_BASIS_POINTS, MAX_BASIS_POINTS - fees.withdrawalFee, Math.Rounding.F
780         : shares;
781 }
```

In contrast, the actual withdrawal logic in `withdraw()` uses `Math.Rounding.Ceil` for the same calculation:

```
648 uint256 feeShares = msg.sender != feeRecipient
649     ? shares.mulDiv(MAX_BASIS_POINTS, MAX_BASIS_POINTS - withdrawalFee, Math.Rounding.Ceil) - sh
650     : 0;
```

As a result, `previewWithdraw()` can underestimate the number of shares required, causing user transactions to revert if they rely on the previewed value.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (2.5)

Recommendation

Align the rounding logic in `previewWithdraw` with the actual withdrawal calculation by using `Math.Rounding.Ceil` for the fee. This ensures previews accurately reflect the required shares for withdrawal.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/sc_earn-v1/commit/509af244fdab377ea04580df004e433cbb1f532c

7.3 REWARD ALLOCATION SUSCEPTIBLE TO SNIPING

// LOW

Description

The `ConcreteMultiStrategyVault` contract distributes rewards to users based on their share balance at the moment the owner calls `harvestRewards()`. The reward calculation from the `RewardsHelper` library uses the current `totalSupply()` of shares:

```
137 | rewardIndex[rewardToken] += amount.mulDiv(PRECISION, totalSupply, Math.Rounding.Floor);
```

A user can monitor the blockchain for a `harvestRewards()` call. Just before this function is executed, the user deposits a large amount, increasing their share of the vault. Immediately after rewards are distributed, the user withdraws their funds. This allows the user to claim a disproportionate share of the rewards, even though their capital did not contribute to generating them. As a result, long-term users receive less than their fair share.

BVSS

AO:A/AC:M/AX:M/R:N/S:U/C:N/A:N/I:N/D:N/Y:M (2.2)

Recommendation

Implement a mechanism to prevent immediate withdrawals after deposits, such as a short lockup period or withdrawal delay, to ensure users cannot game the reward distribution.

Remediation Comment

NOT APPLICABLE: It has been concluded that this issue is not applicable, as stated by the **Blueprint Finance team**:

> *Currently, there are no rewards on the multisig strategy. For general-purpose vaults, here are the measures to avoid sniping:*
> *a.* `harvestRewards()` *can be called frequently to prevent reward accumulation. This works for strategies like Aave and Compound, where rewards are distributed per second or per block.*
> *b. For strategies with infrequent rewards, we have a reward window defined that distributes the reward amount uniformly over a period of time, so* `harvestRewards` *only fetches a share of the rewards and not the entire amount.*

7.4 MISSING RATE CHANGE VALIDATION IN ASSET ADJUSTMENT AFTER UNPAUSE

// LOW

Description

The `MultiSigStrategyV1.unpauseAndAdjustTotalAssets()` function allows the owner to unpause the strategy and immediately adjust the asset accounting by any amount, without enforcing the configured `maxRateChangeThreshold`. The intended protocol safeguard is implemented in `_validateRateChange()`, which calculates the percentage change and reverts or pauses if the change exceeds the threshold. However, `unpauseAndAdjustTotalAssets()` directly calls `_adjustTotalAssets()` and skips this validation.

```
319 function unpauseAndAdjustTotalAssets(int256 diff) external onlyOwner {
320     _unpause();
321     _adjustTotalAssets(diff, getNextErateNonce());
322 }
```

A call to `unpauseAndAdjustTotalAssets()` with a large `diff` value, will instantly update the asset balance by more than the allowed percentage, bypassing the risk controls meant to prevent sudden or suspicious accounting changes.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:N/Y:N (2.0)

Recommendation

Enforce the rate change threshold in `unpauseAndAdjustTotalAssets()` by calling `_validateRateChange()` before updating asset balances, and revert or pause if the change exceeds the configured threshold.

Remediation Comment

RISK ACCEPTED: The **Blueprint Finance team** made a business decision to accept the risk of this finding and not alter the contracts, stating that:

> *The operator performs regular asset updates using `adjustTotalAssets()`, with a maximum rate change threshold in place for safety. If this threshold is breached or the cooldown period is violated, the strategy is paused and flagged for admin review.*
> *We do not enforce strict validation because significant market moves, sudden losses, or short-term profit spikes can legitimately push rate changes beyond the threshold. In such cases, only the admin can approve these off-rate updates after review. The conservative threshold acts as a safeguard to keep the system safe and well-monitored.*

7.5 WITHDRAWAL QUEUE MAXREDEEM DOES NOT ACCOUNT FOR WITHDRAWAL FEE

// INFORMATIONAL

Description

In `ConcreteMultiStrategyVault.maxRedeem()`, when the withdrawal queue is active, the function returns the user's full share balance, without reserving shares for the withdrawal fee or considering available on-chain liquidity. Calling `redeem(maxRedeem)` will burn the full share balance, but net assets delivered will be reduced by the withdrawal fee, and delivery may be queued.

```
610 function maxRedeem(address owner) public view virtual override returns (uint256) {
611     if (paused() || withdrawalsPaused || !_isWhitelisted(owner)) return 0;
612     uint256 userShares = balanceOf(owner);
613     if (address(withdrawalQueue) != address(0)) {
614         return userShares;
615     }
616     uint256 availableAssets = getAvailableAssetsForWithdrawal();
617     uint256 availableAssetsInShares = _convertToShares(availableAssets, Math.Rounding.Floor);
618     return availableAssetsInShares >= userShares ? userShares : availableAssetsInShares;
619 }
```

This causes a minor mismatch between the preview and what users might expect, since `maxRedeem()` under queue mode ignores fees and available liquidity.

BVSS

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

Recommendation

Consider aligning `maxRedeem` with post-fee semantics when the queue is active, or clearly document that `maxRedeem` returns the full user share balance and actual delivery is subject to fee and queue processing.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** made a business decision to acknowledge this finding and not alter the contracts, since the behavior is ERC4626-compliant, stating:

> *As per the ERC-4626 spec: `maxRedeem()` defines the maximum amount of vault shares that can be redeemed (burned) from the owner's balance in the vault through a redeem call:*
> *a. When the queue is active: the user can redeem/burn all shares they hold in the vault without revert.*
> *b. When the queue is inactive: the user can redeem/burn `min(userShares, availableAssets)`.*
> *The `feeShares` are minted to the `feeRecipient` and are not subtracted from the user's share balance, since they are newly minted.*

7.6 CENTRALIZATION RISKS

// INFORMATIONAL

Description

The `ConcreteMultiStrategyVault`, `StrategyBase`, and `MultiSigStrategyV1` contracts grant the `owner`, and in some cases, the `operator`, broad authority over core protocol operations. This includes the ability to pause or unpause the protocol, set or update fee rates and recipients, change deposit limits, manage whitelists and operators, add or remove strategies, adjust strategy allocations, and configure withdrawal infrastructure.

Additionally, in `MultiSigStrategyV1`, normal deposit and withdrawal operations depend on the owner or operator periodically updating the strategy's asset accounting via `adjustTotalAssets()` or `unpauseAndAdjustTotalAssets()`. If the owner/operator fails to perform these updates within the configured validity period, all deposits and withdrawals to the strategy will revert until the admin refreshes the eRate.

As a result, a malicious or compromised owner/operator could arbitrarily disrupt user actions, change fee structures, redirect funds, or otherwise interfere with normal vault operation, potentially harming users and undermining protocol trust.

BVSS

AO:S/AC:L/AX:M/R:N/S:U/C:N/A:H/I:N/D:H/Y:H (1.5)

Recommendation

Document all owner and operator powers and associated risks in the protocol documentation. Consider introducing multi-signature or timelock controls for critical owner actions to reduce single point of failure risks.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** made a business decision to acknowledge this finding and not alter the contracts, stating:

> *To avoid or minimize centralization risks, we set the owner as a multisig safe and the operator as a Fireblocks wallet with defined policies.*

7.7 MISSING VALIDATION FOR REWARD TOKEN AND VAULT FEE VALUES

// INFORMATIONAL

Description

The `modifyRewardFeeForRewardToken()` function in `StrategyBase` contract allows the owner to set the fee for any reward token to an arbitrary value, including 100% (10,000 basis points) or higher.

```
260 function modifyRewardFeeForRewardToken(uint256 newFee_, RewardToken calldata rewardToken_) exter
261     // Ensure the reward token is approved before attempting to modify its fee.
262     if (!rewardTokenApproved[address(rewardToken_.token)]) {
263         revert RewardTokenNotApproved();
264     }
265
266     // Find the index of the reward token to modify.
267     uint256 index = _getIndex(address(rewardToken_.token));
268
269     // Update the fee for the specified reward token.
270     rewardTokens[index].fee = newFee_;
271 }
```

Similarly, the `setVaultFees()` function in `ConcreteMultiStrategyVault` allows the owner to set arbitrary fee values without bounds.

```
906 function setVaultFees(VaultFees calldata newFees_) external takeFees onlyOwner {
907     fees = newFees_; // Update the fee structure
908     feesUpdatedAt = block.timestamp; // Record the time of the fee update
909 }
```

This can result in all rewards being captured as fees, or cause division by zero and revert protocol actions, potentially bricking deposits, withdrawals, or rewards.

BVSS

AO:S/AC:L/AX:M/R:N/S:U/C:N/A:N/I:N/D:C/Y:N (1.3)

Recommendation

Enforce an upper bound on all fee values.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** made a business decision to acknowledge this finding and not alter the contracts.

7.8 NON-STANDARD/FEE-ON-TRANSFER TOKENS CAN INFLATE SHARES

// INFORMATIONAL

Description

The `deposit()` and `mint()` functions from the `ConcreteMultiStrategyVault` contract mint shares based on the `assets_/shares_` arguments, then transfer tokens from the user, but never verify the actual tokens received.

If the asset is a fee-on-transfer or deflationary token, the vault mints shares for more assets than it actually received, resulting in under collateralization and potential loss for other users.

BVSS

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

Recommendation

Forbid fee-on-transfer tokens, or compute `received = balanceAfter − balanceBefore` and use the actual `received` value for mint/share math, and revert if `received < assets_`.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** made a business decision to acknowledge this finding and not alter the contracts, stating that:

> *The current use cases of the multisig strategy are limited to certain blue-chip or standard assets only.*

7.9 FEE BYPASS WHEN FEERECIPIENT IS THE CALLER

// INFORMATIONAL

Description

Entry points apply deposit/withdrawal fees only if `msg.sender != feeRecipient`. This allows the `feeRecipient` to bypass protocol fees by acting as the caller.

For example, if a user approves the `feeRecipient` to spend their shares, the `feeRecipient` can call `withdraw()` or `redeem()` on their behalf and no withdrawal fee is applied. Similarly, the `feeRecipient` can deposit for itself and avoid deposit fees. This allows protocol fee revenue to be bypassed by routing actions through the `feeRecipient`, and can be systematically exploited if users coordinate to avoid fees.

BVSS

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

Recommendation

Remove the `msg.sender` check and always apply the fee when the configured fee is greater than zero. If exemptions are needed, implement an explicit whitelist and check the payer or owner, not the caller.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** made a business decision to acknowledge this finding and not alter the contracts.

7.10 RECEIVER NOT VALIDATED AGAINST WHITELIST

// INFORMATIONAL

Description

The `ConcreteMultiStrategyVault` contract enforces whitelist checks only on the `msg.sender` for deposit, mint, withdraw, and redeem operations, but not on the receiver.

As a result, a non-whitelisted user can receive shares or assets if a whitelisted user acts as a relayer, or can have a whitelisted user withdraw/redeem on their behalf. This undermines the intended access control.

BVSS

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

Recommendation

Require the receiver to be whitelisted when whitelist is enabled.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** made a business decision to acknowledge this finding and not alter the contracts.
