



Accountable Audit Report

Prepared by [Cyfrin](#)

Version 2.1

Lead Auditors

[Immeas](#)

[MrPotatoMagic](#)

January 30, 2026

Contents

1	About Cyfrin	2
2	Disclaimer	2
3	Risk Classification	2
4	Protocol Summary	2
5	Audit Scope	2
6	Executive Summary	3
7	Findings	5
7.1	Low Risk	5
7.1.1	managerSplit can be misconfigured above BASIS_POINTS	5
7.1.2	AccountableOpenTerm::accrueInterest does not refresh delinquency status	5
7.1.3	AccountableAsyncRedeemVault::maxDeposit / maxMint can be stale due to non-accrued scaleFactor breaking EIP-4626 compliancy	5
7.1.4	Protocol fees are not auto-collected before treasury address update	6
7.1.5	Incorrect delinquency status update due to missing interest accrual	6
7.2	Informational	8
7.2.1	Unused errors	8
7.2.2	Unused state variable	8
7.2.3	Unused imports	8
7.2.4	State change without event	9
7.2.5	Prepayment fee is unbounded	9
7.2.6	_updateDelinquentStatus called in both branches in AccountableOpenTerm::repay	9
7.2.7	AccountableOpenTerm-functions incorrectly under View Functions header	9
7.3	Gas Optimization	10
7.3.1	State variables can be immutable	10
7.3.2	Unnecessary FeeManager::managerSplit call	10
7.3.3	Unnecessary check in AccountableAsyncRedeemVault::maxRedeem	10
7.3.4	AccountableAsyncRedeemVault::maxWithdraw can be optimized when state.redeemShares == 0	11
7.3.5	Emit events early to save gas	11

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Accountable is a lending protocol built around ERC-4626-style vaults where lenders deposit into a vault and a borrower draws liquidity via a corresponding loan strategy. The strategy accrues interest and fees over time, enforces capacity/reserve constraints, and supports both fixed-term and open-term loan configurations. Withdrawals are handled asynchronously via a withdrawal queue and optional time-batched processing, so liquidity repaid to the vault is used to progressively satisfy redeem requests.

5 Audit Scope

The audit scope was limited to:

```
// changes since 1ae7e2fb74a3c0f543147e8793785b7f70d25070 to 277d154d9faf9164c6cd32d66cf38f12a73c5087
// see https://github.com/Accountable-Protocol/credit-vaults-internal/compare/1ae7e2fb74a3c0f543147e879
↪ 3785b7f70d25070...277d154d9faf9164c6cd32d66cf38f12a73c5087

// minor changes in `_isVerified`, `_areVerified`
src/access/AccessBase.sol

// minor changes in `_verify`, `_verifyMany`, `_verifySingle`
src/access/Authorizable.sol

// very small change - removed `performanceFeeWaived: params.performanceFeeWaived`
src/factory/FixedTermFactory.sol
src/factory/OpenTermFactory.sol

// very small change - new revert for `InvalidRewardsType`
src/factory/RewardsFactory.sol

// addition of `_hasPrepaymentFee` mapping and various fee-related changes
src/modules/FeeManager.sol
```

```
// minor changes - storage moved into it own file
src/modules/GlobalRegistry.sol
src/modules/storage/GlobalRegistryStorage.sol

// minor changes - remove of `release` function
src/rewards/Rewards.sol

// minor changes in `acceptRoot` and related to claim amounts
src/rewards/RewardsDistributorMerkle.sol

// storage related changes, some new functions and other changes
src/strategies/AccountableFixedTerm.sol
src/strategies/AccountableOpenTerm.sol
src/strategies/AccountableStrategy.sol
src/strategies/storage/FixedTermStorage.sol
src/strategies/storage/OpenTermStorage.sol
src/strategies/storage/StrategyStorage.sol

// bunch of different changes
src/vault/AccountableAsyncRedeemVault.sol
src/vault/AccountableVault.sol
src/vault/queue/AccountableWithdrawalQueue.sol
```

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the [Accountable](#) smart contracts provided by [Accountable](#). In this period, a total of 17 issues were found.

During this follow-up review, we identified five low-risk issues mainly related to configuration and edge-case behavior. These include unbounded fee parameter (e.g., manager split / prepayment fee), view-function inconsistencies where `maxDeposit/maxMint` may be stale due to non-virtual accrual, and a minor state-tracking inconsistency where `accrueInterest()` does not update delinquency status. In addition several informational and gas findings were identified.

After the audit concluded the team added a new commit [62d25bf](#) adding `nonReentrant` to all external calls which was reviewed and deemed to be safe.

Summary

Project Name	Accountable
Repository	credit-vaults-internal
Commit	277d154d9faf...
Fix Commit	809813f5d12c...
Audit Timeline	Jan 19th - Jan 23rd, 2026
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	5
Informational	7
Gas Optimizations	5
Total Issues	17

Summary of Findings

[L-1] <code>managerSplit</code> can be misconfigured above <code>BASIS_POINTS</code>	Resolved
[L-2] <code>AccountableOpenTerm::accrueInterest</code> does not refresh delinquency status	Resolved
[L-3] <code>AccountableAsyncRedeemVault::maxDeposit / maxMint</code> can be stale due to non-accrued <code>scaleFactor</code> breaking EIP-4626 compliancy	Resolved
[L-4] Protocol fees are not auto-collected before treasury address update	Acknowledged
[L-5] Incorrect delinquency status update due to missing interest accrual	Resolved
[I-1] Unused errors	Resolved
[I-2] Unused state variable	Resolved
[I-3] Unused imports	Resolved
[I-4] State change without event	Resolved
[I-5] Prepayment fee is unbounded	Resolved
[I-6] <code>_updateDelinquentStatus</code> called in both branches in <code>AccountableOpenTerm::repay</code>	Resolved
[I-7] <code>AccountableOpenTerm</code> -functions incorrectly under View Functions header	Resolved
[G-1] State variables can be immutable	Resolved
[G-2] Unnecessary <code>FeeManager::managerSplit</code> call	Resolved
[G-3] Unnecessary check in <code>AccountableAsyncRedeemVault::maxRedeem</code>	Resolved
[G-4] <code>AccountableAsyncRedeemVault::maxWithdraw</code> can be optimized when <code>state.redeemShares == 0</code>	Resolved
[G-5] Emit events early to save gas	Resolved

7 Findings

7.1 Low Risk

7.1.1 `managerSplit` can be misconfigured above `BASIS_POINTS`

Description: The `FeeManager` allows setting the manager fee (e.g., `managerSplit`) without enforcing that it is \leq `BASIS_POINTS` (100%). As a result, the manager fee can be set to an invalid value greater than `BASIS_POINTS`.

Impact: If `managerSplit > BASIS_POINTS` is set, fee calculations can become nonsensical or revert. In particular, any logic that derives a complementary “protocol split” as `BASIS_POINTS - managerSplit` may underflow and revert, potentially breaking fee settlement paths and causing operational failures (e.g., inability to collect or distribute fees).

Recommended Mitigation: Add an explicit bounds check when setting the manager fee in `_requireValidFeeStructure`:

```
function _requireValidFeeStructure(FeeStructure memory fees) private pure {  
+   if (fees.managerSplit > BASIS_POINTS) revert InvalidManagerSplit();  
       if (fees.performanceFee > MAX_PERFORMANCE_FEE) revert InvalidPerformanceFee();  
       if (fees.establishmentFee > MAX_ESTABLISHMENT_FEE) revert InvalidEstablishmentFee();  
}
```

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified. `managerSplit` now validated to be below `BASIS_POINTS`.

7.1.2 `AccountableOpenTerm::accrueInterest` does not refresh delinquency status

Description: `AccountableOpenTerm::accrueInterest` only calls `_accrueInterest()` and does not call `_updateDelinquentStatus()`, unlike `updateLateStatus()` which accrues and then updates delinquency state.

Impact: Third parties (keepers/UIs) can advance interest accrual while leaving delinquency state stale until a later call updates it. This can cause temporary inconsistencies in delinquency tracking (e.g., delayed/incorrect `delinquencyStartTime` updates and penalty application timing), which may affect monitoring/automation that relies on delinquency status.

Recommended Mitigation: Either (a) have `accrueInterest()` also call `_updateDelinquentStatus()`, (b) document that keepers should call `updateLateStatus()` when delinquency correctness is required, or (c) remove it and just use `updateLateStatus()` (or vice versa).

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified. `accrueInterest` now calls `_updateDelinquentStatus()`

7.1.3 `AccountableAsyncRedeemVault::maxDeposit / maxMint` can be stale due to non-accrued `scaleFactor` breaking EIP-4626 compliancy

Description: `AccountableOpenTerm.maxDeposit()` computes remaining capacity using `principalAssets = debtShares * _scaleFactor`, but `_scaleFactor` is only updated when `_accrueInterest()` is executed. Since `maxDeposit()` is a view and does not “virtually accrue,” it can return a value based on an outdated `_scaleFactor`. The vault’s `maxMint() / maxDeposit()` inherit this staleness and may report limits that don’t match the current economic state.

Impact: Integrators/users may see `maxDeposit / maxMint` values that are too high, then have `deposit() / mint()` unexpectedly revert once the state-changing path accrues interest and enforces capacity. This is non-compliant with [EIP-4626](#) expectations which states:

MUST return the maximum amount of assets `deposit` would allow to be deposited for receiver and not cause a revert, which MUST NOT be higher than the actual maximum that would be accepted

Recommended Mitigation: Use “virtual accrual” in view functions: compute an up-to-date scale factor for the current timestamp (without writing state) and use it in `maxDeposit` and `share-price/limit` calculations backing `maxMint` (and related `prevIEWS/limits`). Possibly by splitting `_accrueInterest()` into a view and a state changing part.

Accountable: Fixed in commit [5891946](#)

Cyfrin: Verified. `_accrueInterest` now split into `_previewAccruedInterest` which is called from `_accrueInterest`.

7.1.4 Protocol fees are not auto-collected before treasury address update

Description: Function `setTreasury` can be used to update the existing treasury to a new address. However, before updating the treasury address, any existing protocol fees are not auto-collected to the current treasury. While this is not a major issue, it could be problematic if the treasury is updated to a dead or invalid address intentionally or unintentionally.

```
function withdrawProtocolFee(address asset) public nonReentrant onlyTreasury {
    uint256 amount = protocolFees[asset];
    if (amount > 0) {
        protocolFees[asset] = 0;
        IERC20(asset).safeTransfer(treasury, amount);

        emit Withdraw(asset, address(0), treasury, amount);
    }
}

function setTreasury(address treasury_) public onlyOwner {
    if (treasury_ == address(0)) revert ZeroAddress();
    address oldTreasury = treasury;
    treasury = treasury_;
    emit TreasurySet(oldTreasury, treasury_);
}
```

Recommended Mitigation: Consider collecting any protocol fees before updating the treasury address. Additionally, consider adding a two-step transfer when changing the treasury address.

Accountable: Acknowledged. There is no case when we will update the treasury to a dead address such that uncollected fees get lost.

7.1.5 Incorrect delinquency status update due to missing interest accrual

Description: In the `AccountableOpenTerm` contract, the `_scaleFactor` is updated in function `_accrueInterest`. This means that any operation relying on an accurate `_scaleFactor` should accrue interest beforehand.

```
_scaleFactor += baseInterest + delinquencyFee;
```

Function `_calculateRequiredLiquidity` uses the `_scaleFactor` variable to calculate the required reserves, which is then used by function `_isDelinquent` to determine whether the loan is delinquent or not. The boolean value returned from `_isDelinquent` is used by function `_updateDelinquentStatus` to determine the latest status of loan payments. Function `setReserveThreshold` calls `_updateDelinquentStatus` however it does not accrue interest. Due to this, the delinquency status is updated based on a stale `_scaleFactor` value.

```
function setReserveThreshold(uint256 threshold)
    external
    override(AccountableStrategy, IAccountableStrategy)
    onlyManager
{
    if (threshold > BASIS_POINTS) revert ThresholdTooHigh();
    _loan.reserveThreshold = threshold;

    _updateDelinquentStatus();
}
```

```
        emit ReserveThresholdSet(threshold);  
    }  
}
```

Recommended Mitigation: Consider accruing interest before updating the delinquency status.

Accountable: Fixed in commit [809813f](#)

Cyfrin: Verified. `_accrueInterest` now called before `_updateDelinquentStatus` when the loan is ongoing.

7.2 Informational

7.2.1 Unused errors

Description: Consider using or removing the unused errors in `src/constants/Errors.sol`:

- [Line: 40](#)

```
error CancelDepositRequestFailed();
```

- [Line: 67](#)

```
error NoCancelRedeemRequest();
```

- [Line: 79](#)

```
error NoQueueRequests();
```

- [Line: 113](#)

```
error InterestAlreadyClaimed();
```

- [Line: 122](#)

```
error InvalidVaultManager();
```

- [Line: 156](#)

```
error ZeroAmount();
```

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.2.2 Unused state variable

Description: The constant `FeeManager._protocolSplit` is unused. Consider removing or using this unused variable.

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.2.3 Unused imports

Description: Redundant import statements in `src/strategies/AccountableStrategy.sol`. Consider removing them:

- [Line: 6](#)

```
import {RewardsType} from "../interfaces/IRewards.sol";
```

- [Line: 8](#)

```
import {IRewardsFactory} from "../interfaces/IRewardsFactory.sol";
```

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.2.4 State change without event

Description: There is important state changes in this function but no event is emitted. Consider emitting an event to enable offchain indexers to track the changes.

- [AccountableOpenTerm::setProposer](#)

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.2.5 Prepayment fee is unbounded

Description: The prepayment fee can be set without an upper bound. Governance could set an extreme prepayment fee, making early repayment prohibitively expensive or potentially causing unexpected behavior/reverts. This creates user trust and predictability risk.

Consider adding a cap when setting the prepayment fee (e.g., `if(prepaymentFee > MAX_PREPAYMENT_FEE)`).

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.2.6 `_updateDelinquentStatus` called in both branches in `AccountableOpenTerm::repay`

Description: `_updateDelinquentStatus` is called in both branches in `AccountableOpenTerm::repay`

```
if (_loan.outstandingPrincipal == 0) {
    loanState = LoanState.Repaid;
    _updateDelinquentStatus();
} else {
    _updateDelinquentStatus();
}
```

Consider simplifying it to:

```
if (_loan.outstandingPrincipal == 0) {
    loanState = LoanState.Repaid;
}
_updateDelinquentStatus();
```

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.2.7 `AccountableOpenTerm`-functions incorrectly under `View Functions` header

Description: The functions `AccountableOpenTerm::updateLateStatus`, `accrueInterest`, and `processAvailableWithdrawals` are all located under the header:

```
// ===== //
//                               View Functions                               //
// ===== //
```

Consider moving them from there as they are not view functions.

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.3 Gas Optimization

7.3.1 State variables can be immutable

Description: State variables that are only changed in the constructor should be declared immutable to save gas. Add the `immutable` attribute to state variables that are only changed in the constructor

- [AccountableVault.sol#L44](#):

```
IStrategyVaultHooks public strategy;
```

- [AccountableVault.sol#L47](#)

```
uint256 public precision;
```

- [AccessBase.sol#15](#)

```
PermissionLevel public permissionLevel;
```

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.3.2 Unnecessary `FeeManager::managerSplit` call

Description: In `FeeManager::_collectFeeSplit`, if it enters the `else` branch, there's two calls to `managerSplit(strategy)` done:

```
if (managerSplit(strategy) == 0) {
    managerFee = 0;
    protocolFee = amount;
} else {
    managerFee = _split(amount, managerSplit(strategy));
    protocolFee = amount - managerFee;
}
```

This is unnecessary, consider doing just one call:

```
uint256 managerSplit = managerSplit(strategy);
if (managerSplit == 0) {
    managerFee = 0;
    protocolFee = amount;
} else {
    managerFee = _split(amount, managerSplit);
    protocolFee = amount - managerFee;
}
```

Accountable: Fixed in commit [fa6f74c](#)

Cyfrin: Verified.

7.3.3 Unnecessary check in `AccountableAsyncRedeemVault::maxRedeem`

Description: The last check in `AccountableAsyncRedeemVault::maxRedeem` is unnecessary as if it's zero, zero will just be returned

```
function maxRedeem(address controller) public view override returns (uint256 maxShares) {
    VaultState storage state = _vaultStates[controller];
    maxShares = state.redeemShares;
    if (maxShares == 0) return 0; // @audit-issue GAS unnecessary
}
```

Accountable: Fixed in commit [78cd5c7](#)

Cyfrin: Verified.

7.3.4 AccountableAsyncRedeemVault::maxWithdraw can be optimized when state.redeemShares == 0

Description: In AccountableAsyncRedeemVault::maxWithdraw the redeemShares can be done first and save a read if redeemShares is zero:

```
function maxWithdraw(address controller) public view override returns (uint256 maxAssets) {
    VaultState storage state = _vaultStates[controller];
+   if (state.redeemShares == 0) return 0;
    maxAssets = state.maxWithdraw;
-   if (state.redeemShares == 0) return 0;
}
```

Accountable: Fixed in commit [78cd5c7](#)

Cyfrin: Verified.

7.3.5 Emit events early to save gas

Description: Function setTreasury creates a memory variable oldTreasury which is used in the emission of event TreasurySet. However, creating this memory variable is not required if the event is emitted before the state change.

FeeManager.sol

```
function setTreasury(address treasury_) public onlyOwner {
    if (treasury_ == address(0)) revert ZeroAddress();
    address oldTreasury = treasury;
    treasury = treasury_;
    emit TreasurySet(oldTreasury, treasury_);
}
```

AccountableOpenTerm

```
function approveInterestRateChange() external onlyManager {
    uint256 pendingRate_ = pendingInterestRate;

    _accrueInterest();

    _loan.interestRate = pendingRate_;
    delete pendingInterestRate;

    _updateInterestParams();

    emit InterestRateApproved(pendingRate_);
}
```

AccountableStrategy.sol

```
function acceptBorrowerRole() external virtual {
    if (msg.sender != pendingBorrower) revert InvalidPendingBorrower();

    address oldBorrower = borrower;
    borrower = msg.sender;
    pendingBorrower = address(0);

    emit BorrowerChanged(oldBorrower, msg.sender);
}
```

Recommended Mitigation: Consider updating the functions in the following manner

```
function setTreasury(address treasury_) public onlyOwner {  
    if (treasury_ == address(0)) revert ZeroAddress();  
    emit TreasurySet(treasury, treasury_);  
    treasury = treasury_;  
}
```

Accountable: Fixed in commit [78cd5c7](#)

Cyfrin: Verified.