Balmy Earn Audit

Côme du Crest

2024-30-12

Table of contents

Balmy Earn Audit

This document presents the finding of a smart contract audit conducted by Côme du Crest for Gnosis.

Scope

The scope includes all contracts within `Balmy-protocol/earn-core` as of commit c004fa3 and `Balmy-protocol/earn-periphery` as of commit 89267c8.

Context

Earn implements yield generating vaults where users deposit tokens or coins into a strategy and can withdraw them later for a profit. The profit can be made in a different token or coin than the deposited one. Interestingly, the vaults handle partial and complete loss of certain tokens handled by the strategy.

`earn-core` comprises the main earn vault contract responsible for accounting of the deposited or withdrawn assets as well as the strategy registry contract.

`earn-periphery` implements all the strategy functionalities in a very composable and layered fashion. Each strategy is responsible for generating the yield for the deposited tokens for example by connecting to Aave, Compound, or Lido.

Status

The report has been sent to the core developers.

All the issues have been fixed or rightfully acknowledged. Individual commits fixing the issues have been reviewed and responses have been added to the corresponding issues in this report.

Legal Information And Disclaimer

1. This report is based solely on the information provided by Balmy (the “Company”), with the assumption that the information provided to Gnosis is authentic, accurate, complete, and not misleading as of the date of this report. Gnosis has not conducted any independent enquiries, investigations or due diligence in respect of the Company, its business or its operations.
2. Changes to the information contained in the documents, repositories and any other materials referenced in this report might affect or change the analysis and conclusions presented. Gnosis is not responsible for monitoring, nor will we be aware of, any future additions, modifications, or deletions to the audited code. As such, Gnosis does not assume any responsibility to update any information, content or data contained in this report following the date of its publication.
3. This report does not address, nor should it be interpreted as addressing, any regulatory, tax or legal matters, including but not limited to: tax treatment, tax consequences, levies, duties, data privacy, data protection laws, issues relating to the licensing of information technology, intellectual property, money laundering and countering the financing of terrorism, or any other legal restrictions or prohibitions. Gnosis disclaims any liability for omissions or errors in the findings or conclusions presented in this report.
4. The views expressed in this report are solely our views regarding the specific issues discussed within this report. This report is not intended to be exhaustive, nor should it be construed as an assurance, guarantee or warranty that the code is free from bugs, vulnerabilities, defects or deficiencies. Different use cases may carry different risks, and integration with third-party applications may introduce additional risks.
5. This report is provided for informational purposes only and should not be used as the basis for making investment or financial decisions. This report does not constitute investment research and should not be viewed as an invitation, recommendation, solicitation or offer to subscribe for or purchase any securities, investments, products or services. Gnosis is not a financial advisor, and this report does not constitute financial or investment advice.
6. The statements in this report should be considered as a whole, and no individual statement should be extracted or referenced independently.
7. This report is addressed exclusively to the Company, and except with prior written consent from Gnosis, it may not be shared, disclosed, transmitted, or relied upon by any other person or entity for any purpose.
8. To the fullest extent permitted by applicable laws, Gnosis disclaims any and all other liability, whether in contract, tort, or otherwise, that may arise from this report or the use thereof.

Issues

[High] Performance fees are charged on deposit of vault shares

Summary

When a user deposits into a strategy using the vault yield-bearing token of the strategy instead of its underlying asset (e.g. aToken instead of Token for Aave), the performance fee calculation will consider the newly deposited assets as acquired and charge a fee on them.

Vulnerability Detail

Upon deposits the `EarnVault` directly transfers the deposited tokens from the user to the strategy before calling `strategy.deposited()`:

```
    function _increasePosition(
        uint256 positionId,
        StrategyId strategyId,
        IEarnStrategy strategy,
        uint256 totalShares,
        uint256 positionAssetBalance,
        uint256 positionShares,
        address[] memory tokens,
        CalculatedDataForToken[] memory calculatedData,
        uint256[] memory totalBalances,
        address depositToken,
        uint256 depositAmount
    ) internal returns (uint256 depositedAmount, uint256 assetsDeposited) {
        ...

        depositToken.transferIfNativeOrTransferFromIfERC20({ recipient: address
            (strategy), amount: depositedAmount });
        assetsDeposited = strategy.deposited(depositToken, depositedAmount);

        ...
    }
```

The strategy's fee calculation is called and charges a performance fee on the current balance of the underlying asset returned by `_fees_underlying_totalBalances()`:

```
    function _fees_deposited(
        address depositToken,
        uint256 depositAmount
    ) internal override returns (uint256 assetsDeposited) {
        Fees memory fees = _getFees();
        if (fees.performanceFee == 0) {
            // If performance fee is 0, we will need to clear the last balance.
                Otherwise, once it's turned on again,
            // we won't be able to understand difference between balance changes and yield
            address asset = _fees_underlying_asset();
            _clearBalanceIfSet(asset);
            return _fees_underlying_deposited(depositToken, depositAmount);
        }

        // Note: we are only updating fees for the asset, since it's the only token whose balance will change
        (address[] memory tokens, uint256[] memory currentBalances) = _fees_underlying_totalBalances();  // @audit balance may have already increased
        uint256 performanceFees = _calculateFees(tokens[0], currentBalances[0], fees.performanceFee);

        assetsDeposited = _fees_underlying_deposited(depositToken, depositAmount);

        _performanceData[tokens[0]] = PerformanceData({
            // Note: there might be a small wei difference here, but we can ignore it since it should be negligible
            lastBalance: (currentBalances[0] + assetsDeposited).toUint128(),
            performanceFees: performanceFees.toUint120(),
            isSet: true
        });
    }
```

For example for AaveV3Connector, the balance of the asset is equal to the balance of the vault shares which has already been increased by the depositing user:

```
    function _connector_totalBalances()
        internal
        view
        override
        returns (address[] memory tokens, uint256[] memory balances)
    {
        IERC20 vault_ = aToken();
        IAaveV3Rewards rewards_ = rewards();
        tokens = _connector_allTokens();
        balances = new uint256[](tokens.length);
        balances[0] = vault_.balanceOf(address(this));
        address[] memory asset = new address[](1);
        asset[0] = address(vault_);
        ...
    }
```

Impact

Performance fees are charged on the deposited vault shares while they should only be charged on the shares accrued after the user's deposit. This issue affects every implemented strategies.

Code Snippets

https://github.com/Balmy-protocol/earn-core/blob/c004fa3f7522d9231bb8e36113e32634608ccf38/src/vault/EarnVault.sol#L554-L601

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/layers/fees/external/ExternalFees.sol#L153-L182

Recommendation

If the deposited token is the vault token, increase the `PerformanceData.lastBalance` accordingly before applying the performance fees. Alternatively, notify the strategy before transferring tokens to it for accounting and after transferring tokens for handling the tokens.

Response

Fixed by not transferring the tokens to the strategy and instead having the strategy take the tokens from the vault. This allowed the fee layer to make calculations before the tokens are transferred.

The commit responsible for not directly transferring tokens to the strategy on `earn-core` is `7a58a70`. The fees are then being withdrawn from the vault by the strategy after the fees calculation in commit `2ef7bcd` of `earn-periphery`.

[Med] LiquidityMiningManager.setCampaign() can be re-entered to break accounting

Summary

The function `LiquidityMiningManager.setCampaign()` used to set up a liquidity mining campaign for a strategy can be re-entered by the `MANAGE_CAMPAIGNS_ROLE` to break the accounting of `campaign.pendingFromLastUpdate` which impacts the global accounting of the strategy through the calculation of the reward amounts.

Vulnerability Detail

The function `setCampaign()` can be used by the campaign manager to update a campaign for example to lower the emission rate and thus the total rewards. In that case, the contract will refund tokens (or native coins) provisioned for the campaign back to the sender. The sender can re-enter the function upon receiving coins:

```
    function setCampaign(
        StrategyId strategyId,
        address reward,
        uint256 emissionPerSecond,
        uint256 duration
    ) external payable override onlyRole(MANAGE_CAMPAIGNS_ROLE) {
        bytes32 key = _key(strategyId, reward);
        Campaign storage campaign = _campaigns[key];
        Campaign memory campaignMem = campaign;

        if (campaignMem.lastUpdated == 0) {
            IEarnStrategy strategy = STRATEGY_REGISTRY.getStrategy(strategyId);
            if (strategy.asset() == reward) {
                revert InvalidReward();
            }
            _rewards[strategyId].push(reward);
        } else {
            // Update the pending rewards
            campaign.pendingFromLastUpdate = (_calculateRewardAmount(campaign))
                .toUint104();
        }
        uint256 deadline = block.timestamp + duration;
        uint256 balanceNeeded = emissionPerSecond * duration;
        uint256 currentBalance = (campaignMem.deadline > block.timestamp)
            ? campaignMem.emissionPerSecond * (campaignMem.deadline - block.
                timestamp)
            : 0;
        if (currentBalance < balanceNeeded) {
            uint256 missing = balanceNeeded - currentBalance;
            if (reward == Token.NATIVE_TOKEN) {
                if (msg.value < missing) {
                    revert InsufficientBalance();
                }
                // Return the excess tokens
                if (msg.value > missing) {
                    Token.NATIVE_TOKEN.transfer(msg.sender, msg.value - missing);  // @audit re-enter
                }
            } else {
                // Transfer the missing tokens
                IERC20(reward).safeTransferFrom(msg.sender, address(this), missing);
            }
        } else if (currentBalance > balanceNeeded) {
            // Return the excess tokens
            // slither-disable-next-line arbitrary-send-eth,reentrancy-eth,reentrancy-events,reentrancy-unlimited-gas
            reward.transfer({ recipient: msg.sender, amount: currentBalance - balanceNeeded });
        }

        campaign.emissionPerSecond = emissionPerSecond.toUint88();
        campaign.deadline = deadline.toUint32();
        campaign.lastUpdated = block.timestamp.toUint32();
        emit CampaignSet(strategyId, reward, emissionPerSecond, deadline);
    }
```

When this function is re-entered it updates the storage value `campaign.pendingFromLastUpdate` via `_calculateRewardAmount()` which always increases the value if the campaign is not over and has not been updated in the current block:

```
    function _calculateRewardAmount(Campaign memory campaign) internal view returns (uint256) {
        return
            campaign.pendingFromLastUpdate +
            (
                campaign.lastUpdated < campaign.deadline
                    ? campaign.emissionPerSecond * (Math.min(block.timestamp, campaign.deadline) - campaign.lastUpdated)
                    : 0
            );
    }
```

As a result, the value `campaign.pendingFromLastUpdate` can be pumped up to any value. The function `rewardAmount()` will re-use this inflated value to calculate the rewards for the managed strategy.

```
    function rewardAmount(StrategyId strategyId, address token) external view override returns (uint256) {
        Campaign memory campaign = _campaigns[_key(strategyId, token)];
        return _calculateRewardAmount(campaign);
    }
```

The `rewardAmount()` function is later used by the strategy itself for its own accounting of its total balances:

```solidity
    function _liquidity_mining_totalBalances()
        internal
        view
        override
        returns (address[] memory tokens, uint256[] memory balances)
    {
        (
            address[] memory underlyingTokens,
            uint256[] memory underlyingBalances
        ) = _liquidity_mining_underlying_totalBalances();
        (tokens, balances) = _combineArraysWithRewards(underlyingTokens,
           underlyingBalances);
    }

    function _combineArraysWithRewards(
        address[] memory underlyingTokens,
        uint256[] memory underlyingAmounts
    ) private view returns (address[] memory tokens, uint256[] memory amounts)
       {
        StrategyId strategyId_ = strategyId();
        ILiquidityMiningManagerCore manager = _getLiquidityMiningManager();
        address[] memory rewardsTokens = manager.rewards(strategyId_);
        tokens = new address[](underlyingTokens.length + rewardsTokens.length);
        amounts = new uint256[](underlyingTokens.length + rewardsTokens.length)
            ;
        for (uint256 i; i < underlyingTokens.length; ++i) {
            tokens[i] = underlyingTokens[i];
            amounts[i] = underlyingAmounts[i];
        }
        uint256 tokensIndex = underlyingTokens.length;
        for (uint256 i; i < rewardsTokens.length; ++i) {
            address rewardToken = rewardsTokens[i];
            uint256 rewardAmount = manager.rewardAmount(strategyId_,
               rewardToken);  // @audit possibly inflated value
            (bool isRepeated, uint256 indexRepeated) = _isRepeated(rewardToken,
                underlyingTokens);
            if (isRepeated) {
                amounts[indexRepeated] += rewardAmount;
            } else {
                tokens[tokensIndex] = rewardToken;
                amounts[tokensIndex++] = rewardAmount;
            }
        }
        ...
    }
```

Impact

The `MANAGE_CAMPAIGNS_ROLE` is able to break the accounting of managed strategies resulting in an inflated balance for the strategy. This can be abused to withdraw more funds than should be possible from a strategy. This can also brick the strategy by inflating its balance past what could be withdrawn or by withdrawing all funds from the `LiquidityMiningManager`.

The severity of the issue depends on the trust put in the campaign manager role.

Code Snippets

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/layers/liquidity-mining/external/LiquidityMiningManager.sol#L126-L180

Recommendation

Prevent re-entrancy by reimbursing the manager at the end of the `setCampaign()` function or using a re-entrancy lock.

Response

Fixed by applying the check-effect-interaction pattern as of commit `6b5bdce`.

[Med] Aave and Compound Special withdraws do not grant rewards

Summary

The special withdraws for Aave and Compound connectors do not grant rewards associated with the underlying yield generating token.

Vulnerability Detail

The function `_connector_specialWithdraw()` of `AaveV3Connector` does not claim pending rewards or send any accumulated reward tokens to the recipient:

```
    function _connector_specialWithdraw(
        uint256,
        SpecialWithdrawalCode withdrawalCode,
        uint256[] calldata toWithdraw,
        bytes calldata,
        address recipient
    )
        internal
        override
        returns (
            uint256[] memory balanceChanges,
            address[] memory actualWithdrawnTokens,
            uint256[] memory actualWithdrawnAmounts,
            bytes memory result
        )
    {
        if (
            withdrawalCode == SpecialWithdrawal.
                WITHDRAW_ASSET_FARM_TOKEN_BY_AMOUNT ||
            withdrawalCode == SpecialWithdrawal.
                WITHDRAW_ASSET_FARM_TOKEN_BY_ASSET_AMOUNT
        ) {
            IERC20 aaveVault = aToken();
            balanceChanges = new uint256[](_connector_allTokens().length);
            actualWithdrawnTokens = new address[](1);
            actualWithdrawnAmounts = new uint256[](1);
            result = "";
            uint256 assets = toWithdraw[0];
            aaveVault.safeTransfer(recipient, assets);
            balanceChanges[0] = assets;
            actualWithdrawnTokens[0] = address(aaveVault);
            actualWithdrawnAmounts[0] = assets;
        } else {
            revert InvalidSpecialWithdrawalCode(withdrawalCode);
        }
    }
```

Similarly, the function `_connector_specialWithdraw()` of `CompoundV2Connector` does not claim and transfer `COMP` tokens:

```
    function _connector_specialWithdraw(
        uint256,
        SpecialWithdrawalCode withdrawalCode,
        uint256[] calldata toWithdraw,
        bytes calldata,
        address recipient
    )
        internal
        virtual
        override
        returns (
            uint256[] memory balanceChanges,
            address[] memory actualWithdrawnTokens,
            uint256[] memory actualWithdrawnAmounts,
            bytes memory result
        )
    {
        ICERC20 cToken_ = cToken();
        balanceChanges = new uint256[](2);

        actualWithdrawnTokens = new address[](1);
        actualWithdrawnAmounts = new uint256[](1);
        result = "";
        if (withdrawalCode == SpecialWithdrawal.
           WITHDRAW_ASSET_FARM_TOKEN_BY_AMOUNT) {
            uint256 shares = toWithdraw[0];
            uint256 balanceBefore = cToken_.viewUnderlyingBalanceOf(address(
                this));
            IERC20(cToken_).safeTransfer(recipient, shares);
            uint256 balanceAfter = cToken_.viewUnderlyingBalanceOf(address(this
                ));
            balanceChanges[0] = balanceBefore - balanceAfter;
            actualWithdrawnTokens[0] = address(cToken_);
            actualWithdrawnAmounts[0] = shares;
        } else if (withdrawalCode == SpecialWithdrawal.
           WITHDRAW_ASSET_FARM_TOKEN_BY_ASSET_AMOUNT) {
            // Note: we round down because if we were to round up, we might end
                 up withdrawing more than the position's
            // balance, which would end up reverting on the vault
            uint256 shares = _convertAssetsToShares(toWithdraw[0], Math.
                Rounding.Floor);
            uint256 balanceBefore = cToken_.viewUnderlyingBalanceOf(address(
                this));
            IERC20(cToken_).safeTransfer(recipient, shares);
            uint256 balanceAfter = cToken_.viewUnderlyingBalanceOf(address(this
                ));
            balanceChanges[0] = balanceBefore - balanceAfter;
            actualWithdrawnTokens[0] = address(cToken_);
            actualWithdrawnAmounts[0] = shares;
        } else {
            revert InvalidSpecialWithdrawalCode(withdrawalCode);
        }
    }
```

Impact

As far as my understanding goes, the rewards from Aave and Compound are tied to the address providing the tokens and solely transferring the aToken or cToken will not result in a transfer of the underlying rewards.

The rewards gathered by the strategy will not be distributed to the user of special withdraws.

Code Snippets

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/layers/connector/AaveV3Connector.sol#L244-L277

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/layers/connector/compound-v2/CompoundV2Connector.sol#L235-L279

Recommendation

Claim all pending rewards in special withdraws and send rightfully accumulated rewards to the recipient. This requires updating and distributing accumulated reward before the withdraw occurs which requires vault interaction.

Response

This is intended. For now, special withdrawals are meant to be used to withdraw the asset or share token only. Both these options would only affect the user's asset balance, not the rewards balance. Those can be withdrawn by the user later, using the withdraw function.

[Med] AaveV3 connector does not migrate rewards

Summary

The AaveV3 connector is not migrating rewards in `_connector_migrateToNewStrategy()` meaning that rewards will be lost during a strategy migration.

Vulnerability Detail

The function `_connector_migrateToNewStrategy()` does not migrate any of the rewards tokens:

```
    function _connector_migrateToNewStrategy(
        IEarnStrategy newStrategy,
        bytes calldata
    ) internal override returns (bytes memory) {
        IERC20 vault_ = aToken();
        uint256 balance = vault_.balanceOf(address(this));
        vault_.safeTransfer(address(newStrategy), balance);
        return abi.encode(balance);
    }
```

Impact

As far as my understanding goes, the rewards from Aave are tied to the address providing the tokens and solely transferring the aToken will not result in a transfer of the underlying rewards.

During a migration from one strategy to the other, rewards will not be migrated and will be lost forever.

Code Snippets

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/layers/connector/AaveV3Connector.sol#L280-L292

Recommendation

Withdraw and migrate rewards when migrating an AaveV3 strategy.

Response

Rewards are now collected and migrated as of commit `8ddde37`.

[Low] Beefy connector may run out of shares to withdraw

Summary

The Beefy connector rounds up the amount of shares to withdraw when attempting to withdraw assets from the beefy vault. If a user attempts to remove all the assets held by the strategy this could result in a number of shares than the number of shares held by the strategy. The resulting withdraw will fail.

Vulnerability Detail

In `BeefyConnector _connector_withdraw()` uses `_convertAssetsToShares(vault, assets)` to calculate the number of shares to withdraw:

```
    function _connector_withdraw(
        uint256,
        address[] memory,
        uint256[] memory toWithdraw,
        address recipient
    ) internal override returns (IEarnStrategy.WithdrawalType[] memory) {
        uint256 assets = toWithdraw[0];
        IBeefyVault vault = beefyVault();
        IERC20 asset = _asset();

        // We convert the assets to shares with rounding up
        // This way we make sure that the correct amount is withdrawn
        uint256 shares = _convertAssetsToShares(vault, assets);  // @audit we
            may run out of shares to withdraw
        vault.withdraw(shares);

        ...
    }
```

The function `_convertAssetsToShares()` rounds upward the number of calculated shares:

```
    function _convertAssetsToShares(IBeefyVault vault, uint256 assets) private
        view returns (uint256) {
        uint256 totalSupply = vault.totalSupply();
        if (totalSupply == 0) {
            return assets;
        }
        return assets.mulDiv(totalSupply, vault.balance(), Math.Rounding.Ceil);
    }
```

The function `vault.withdraw()` attempts to burn the exact amount of shares passed as argument and will revert when not enough shares are held by the sender:

```
function withdraw(uint256 _shares) public {
    uint256 r = (balance() * _shares) / totalSupply();
    _burn(msg.sender, _shares);
    ...
}
```

Impact

A user attempting to withdraw all the remaining assets of a strategy will likely see the transaction reverting due to the rounding up of the number of shares to withdraw.

Code Snippets

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/layers/connector/BeefyConnector.sol#L156-L184

https://docs.beefy.finance/developer-documentation/vault-contract#withdraw

Recommendation

Round down the amount of shares to withdraw or limit the amount of shares to withdraw by the balance of the strategy.

Response

Fixed by rounding down the amount of shares to withdraw in commit `117f686`.

[Low] Rounding in disadvantage of strategy in BeefyConnector

Summary

The roundings calculated in `BeefyConnector` are often to the disadvantage of the strategy and advantage of the user.

Vulnerability Detail

In `_connector_specialWithdraw()` if the user attempts to withdraw farm token by asset amount, the number of shares withdrawn in the name of user is rounded up. On the other hand if the user attempts to withdraw farm token by amount, the value of `balancesChanges[0]` is the rounded down conversion from withdrawn shares to asset:

```solidity
    function _connector_specialWithdraw(
        uint256,
        SpecialWithdrawalCode withdrawalCode,
        uint256[] calldata toWithdraw,
        bytes calldata,
        address recipient
    )
        internal
        override
        returns (
            uint256[] memory balanceChanges,
            address[] memory actualWithdrawnTokens,
            uint256[] memory actualWithdrawnAmounts,
            bytes memory result
        )
    {
        IBeefyVault vault = beefyVault();
        balanceChanges = new uint256[](1);
        actualWithdrawnTokens = new address[](1);
        actualWithdrawnAmounts = new uint256[](1);
        result = "";
        if (withdrawalCode == SpecialWithdrawal.
           WITHDRAW_ASSET_FARM_TOKEN_BY_AMOUNT) {
            uint256 shares = toWithdraw[0];
            uint256 assets = _convertSharesToAssets(vault, shares);
            vault.safeTransfer(recipient, shares);
            balanceChanges[0] = assets;  // @audit rounding down the amount of
                assets withdrawn by users
            actualWithdrawnTokens[0] = address(vault);
            actualWithdrawnAmounts[0] = shares;
        } else if (withdrawalCode == SpecialWithdrawal.
           WITHDRAW_ASSET_FARM_TOKEN_BY_ASSET_AMOUNT) {
            uint256 assets = toWithdraw[0];
            uint256 shares = _convertAssetsToShares(vault, assets);  // @audit
                rounding in disadvantage of vault
            vault.safeTransfer(recipient, shares);
```

```solidity
                balanceChanges[0] = assets;
                actualWithdrawnTokens[0] = address(vault);
                actualWithdrawnAmounts[0] = shares;
            } else {
                revert InvalidSpecialWithdrawalCode(withdrawalCode);
            }
        }

        function _convertSharesToAssets(IBeefyVault vault, uint256 shares) private
            view returns (uint256) {
            uint256 totalSupply = vault.totalSupply();
            if (totalSupply == 0) {
                return shares;
            }
            return shares.mulDiv(vault.balance(), totalSupply, Math.Rounding.Floor)
                ;
        }

        function _convertAssetsToShares(IBeefyVault vault, uint256 assets) private
            view returns (uint256) {
            uint256 totalSupply = vault.totalSupply();
            if (totalSupply == 0) {
                return assets;
            }
            return assets.mulDiv(totalSupply, vault.balance(), Math.Rounding.Ceil);
        }
```

This value of `balancesChanges` is used by the `EarnVault` to account for the number of assets the user has withdrawn in `EarnVault.specialWithdraw()`:

```solidity
        function specialWithdraw(
            uint256 positionId,
            SpecialWithdrawalCode withdrawalCode,
            uint256[] calldata toWithdraw,
            bytes calldata withdrawalData,
            address recipient
        )
            public
            payable
            virtual
            onlyWithPermission(positionId, WITHDRAW_PERMISSION)
            nonReentrant
            returns (
                address[] memory tokens,
                uint256[] memory balanceChanges,
                address[] memory actualWithdrawnTokens,
                uint256[] memory actualWithdrawnAmounts,
                bytes memory result
            )
        {
            ...

            // slither-disable-next-line reentrancy-no-eth
            (balanceChanges, actualWithdrawnTokens, actualWithdrawnAmounts, result)
                 = strategy.specialWithdraw({
```

```
                positionId: positionId,
                withdrawalCode: withdrawalCode,
                toWithdraw: toWithdraw,
                withdrawalData: withdrawalData,
                recipient: recipient
            });
            // slither-disable-next-line unused-return
            (, uint256[] memory balancesAfterUpdate) = strategy.totalBalances();

            _updateAccounting({
                positionId: positionId,
                strategyId: strategyId,
                totalShares: totalShares,
                positionShares: positionShares,
                positionAssetBalanceBeforeUpdate: positionAssetBalance,
                tokens: tokens_,
                calculatedData: calculatedData,
                balancesBeforeUpdate: balancesBeforeUpdate,
                updateAmounts: balanceChanges,  // @audit lowers user balance by
                    balanceChanges
                balancesAfterUpdate: balancesAfterUpdate,
                action: UpdateAction.WITHDRAW
            });

            ...
        }
```

That means the balance of the user will be updated by a rounded down value.

Impact

The strategy may not old enough beefy shares to cover its total accounted balance in the vault after rounded withdrawals. This will prevent total withdrawal of shares by the strategy user.

Code Snippets

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/layers/connector/BeefyConnector.sol#L187-L225

Recommendation

Round the amount of shares withdrawn from the beefy vault down to ensure strategy is left with enough shares to cover all its accounted assets.

Response

One of the rounding has been fixed in commit `117f686`. The other rounding has been fixed in commit `99ae252`.

[Info] Strategies do not init ExternalLiquidityMining

Summary

Implemented strategies inherit from `ExternalLiquidityMining` but do not call `_liquidity_mining_init()` on initialisation.

Vulnerability Detail

`AaveV3Strategy` for example inherits from `ExternalLiquidityMining` but does not call `_liquidity_mining_init()` on initialisation:

```
contract AaveV3Strategy is
    BaseStrategy,
    Clone,
    AaveV3Connector,
    ExternalLiquidityMining,
    ExternalFees,
    ExternalGuardian,
    ExternalCreationValidation
{
    ...

    function init(
        bytes calldata creationValidationData,
        bytes calldata guardianData,
        bytes calldata feesData,
        string calldata description_
    ) public initializer {
        _creationValidation_init(creationValidationData);
        _guardian_init(guardianData);
        _fees_init(feesData);
        _connector_init();
        description = description_;
    }

    ...
}
```

The liquidity mining init is responsible for configuring the strategy on the manager:

```
    function _liquidity_mining_init(bytes calldata data) internal
       onlyInitializing {
        ILiquidityMiningManagerCore manager = _getLiquidityMiningManager();
        manager.strategySelfConfigure(data);
    }
```

Luckily, the corresponding function of the manager is currently not doing anything:

```
    function strategySelfConfigure(bytes calldata data) external override {
        // Does nothing, we want to have this function for future liquidity
            mining manager implementations
    }
```

This is also the case for `CompoundV2Strategy`, `ERC4626Strategy`, and `LidoSTETHStrategy`
.

Impact

This does not currently cause any issues because the initialisation is not strictly necessary but could cause problems in the future if liquidity mining (or its manager) is updated.

Code Snippets

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/instances/aave-v3/AaveV3Strategy.sol#L58-L72

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/instances/compound-v2/CompoundV2Strategy.sol#L61-L75

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/instances/erc4626/ERC4626Strategy.sol#L57-L71

https://github.com/Balmy-protocol/earn-periphery/blob/89267c868f425c404856847775d5deeef46b48f1/src/strategies/instances/lido/LidoSTETHStrategy.sol#L53-L64

Recommendation

Call `_liquidity_mining_init()` on initialisation of strategies to avoid future update mistakes.

Response

Fixed in commit `7f08482` by calling `_liquidity_mining_init()` in the strategies initialisation.

Optimisations and miscellaneous

This part lists minor gas/code optimizations that shouldn't make the code less readable or improve overall readability. It also lists questions about unclear code segments.

NFTPermissions silently fails on same block updates

NFTPermissions inherited by EarnVault silently fails if a user attempts to set permissions on the same block (timestamp) a token is transferred. I.e. if I transfer a position token to someone else and that person sets custom permissions right after (in the same tx or same block) the execution will not revert but the permissions will be ignored. It is not a security issue but rather an opinion that this is missleading and could lead to unexpected behaviour. https://github.com/Balmy-protocol/nft-permissions/blob/main/src/NFTPermissions.sol

Response: fixed in commit `706d9cc`

Typos

Typo in `YieldDataForTokenLibrary` using `yieldLossData` instead of `yieldData`.

https://github.com/Balmy-protocol/earn-core/blob/c004fa3f7522d9231bb8e36113e32634608ccf38/src/vault/types/YieldDataForToken.sol#L71

https://github.com/Balmy-protocol/earn-core/blob/c004fa3f7522d9231bb8e36113e32634608ccf38/src/vault/types/YieldDataForToken.sol#L83

Response: fixed in commit `4f2eca8`