SHERLOCK

Security Review For Beraborrow

Collaborative Audit Prepared For:	**Beraborrow**
Lead Security Expert(s):	**0x73696d616f** **pkqs90**
Date Audited:	**April 25 - April 30, 2025**
Final Commit:	**8c2f085**

Introduction

TBA

Scope

Repository: Beraborrowofficial/blockend

Audited Commit: 8906ef180a2b0bb9fdb4e32d93e45819b4e540cb

Final Commit: 8c2f085451a02aba9c6998ac84d3e21a13f3fdb9

Files:

- src/core/boyco/ManagedLeveragedVault.sol
- src/dependencies/BeraborrowMath.sol
- src/interfaces/core/IBorrowerOperations.sol
- src/interfaces/core/ILiquidStabilityPool.sol
- src/interfaces/core/IMetaBeraborrowCore.sol
- src/interfaces/core/IPriceFeed.sol
- src/interfaces/core/vaults/IInfraredCollateralVault.sol
- src/interfaces/periphery/ICollVaultRouter.sol
- src/libraries/EmissionsLib.sol
- src/libraries/PriceLib.sol

Final Commit Hash

8c2f085451a02aba9c6998ac84d3e21a13f3fdb9

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
5	13	13

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Lost withdrawals due to ManagedLeveraged Vault::openDen() using all asset() balance

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/22

Summary

`ManagedLeveragedVault::openDen()` picks up all `asset()` and converts to collateral. However, some of these assets may be reserved for withdrawals, in which case the tracking would be completely off.

Vulnerability Detail

`ManagedLeveragedVault::executeWithdrawalEpoch()` converts collateral vault shares to assets and leaves them in the contract.

```
function executeWithdrawalEpoch(
    ExecuteWithdrawalParams calldata params
) external onlyOwner nonReentrant claimCollateralSurplus(params.upperHint,
↪  params.lowerHint) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    _checkEpoch(params.epoch);

    CollDebt memory cd = _getCollVaultSharesAndDebtToUnwind(params.epoch);

    b$.borrowerOperations.adjustDen({
        denManager: b$.denManager,
        account: address(this),
        _maxFeePercentage: params.maxFeePercentage,
        _collDeposit: 0,
        _collWithdrawal: cd.collVaultSharesToWithdraw,
        _debtChange: cd.debtToUnwind,
        _isDebtIncrease: false,
        _upperHint: params.upperHint,
        _lowerHint: params.lowerHint
    });

    /// @dev Have assets sitting on the contract for `withdrawFromEpoch` to claim
    uint256 assets = _rebalance(
            RebalanceParams({
                sentCurrency: address(b$.collVault),
                sentAmount: cd.collVaultSharesToWithdraw,
                receivedCurrency: asset(),
                ext: params.unwrapParams
            })
```

```
        );
    ...
}
```

`ManagedLeveragedVault::openDen()` picks up all `asset()` available, including the ones reserved for withdrawal.

```
function openDen(
    uint256 _maxFeePercentage,
    uint256 _minCollVaultShares,
    address _upperHint,
    address _lowerHint
) external onlyOwner {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    IDenManager denManager = IDenManager(b$.denManager);

    uint256 status = denManager.getDenStatus(address(this));
    if (DenStatus(status) != DenStatus.active) revert AlreadyOpened();

    uint256 collVaultShares;
    uint256 assetsAmount = IERC20(asset()).balanceOf(address(this));
    if (assetsAmount != 0) {
        IERC20(asset()).approve(address(b$.collVault), assetsAmount);
        collVaultShares = b$.collVault.deposit(assetsAmount, address(this));
        if (collVaultShares < _minCollVaultShares) revert
↪   VaultSlippage(_minCollVaultShares, collVaultShares);
    }
    ...
}
```

Impact

DoSed withdrawals, loss of funds as the shares/assets ratio will be off because the shares were already burned.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR521

Tool Used

Manual Review

Recommendation

Include the reserved assets for withdrawal in the calculation.

Discussion

alex-beraborrow

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1 allowbreak #discussion_r2066805203

Issue H-2: Users cannot withdraw correctly when exposure token is sNECT.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/29

Summary

Users cannot withdraw correctly when exposure token is sNECT.

Vulnerability Detail

When withdraw is being handled, `_getCollVaultSharesAndDebtToUnwind()` function is called to unwind the exposure token to NECT to repay debt. There are two cases, for the exposure token:

1. Exposure token is sNECT. We need to do `LSP.withdraw()` to convert sNECT -> NECT.
2. Exposure token is bb.sNECT. We need to first do `CollateralVault.withdraw()` to convert bb.sNECT -> sNECT, then `LSP.withdraw()` to convert sNECT -> NECT.

The issue is with case 1. When calling `IERC4626($.exposureToken).withdraw`, we should pass in the amount of `cd.debtToUnwind` instead of `lspNeeded`, because the parameter passed in should be asset amount instead of share amount.

This will result in less NECT withdrawn, which is charged from withdrawers, causing loss of funds.

```
    function _getCollVaultSharesAndDebtToUnwind(uint256 _epoch) internal returns
↪   (CollDebt memory cd) {
        ...
        uint256 lspNeeded = IERC4626(b$.lsp).previewWithdraw(cd.debtToUnwind);
@>      uint256 exposureWithdrawn = IERC4626($.exposureToken).withdraw(lspNeeded,
↪   address(this), address(this));
        if ($.exposureToken != address(b$.lsp)) {
            // LSP is asset of BB.SNECT
            IERC4626(b$.lsp).withdraw(cd.debtToUnwind, address(this),
↪   address(this));
        }

        cd.collVaultSharesToWithdraw =
↪   getCollVaultSharesToWithdraw(collVaultSharesRequested, cd.debtToUnwind,
↪   exposureWithdrawn);

        collVaultSharesRequested = (collVaultSharesRequested * (BP -
↪   $.maxWithdrawalLossInBP)) / BP;
        if (cd.collVaultSharesToWithdraw < collVaultSharesRequested) revert
↪   VaultSlippage(cd.collVaultSharesToWithdraw, collVaultSharesRequested);
```

```
    }
```

Impact

Loss of funds for withdrawers.

Recommendation

Fix the logic so we withdraw the correct amount of NECT.

Discussion

alex-beraborrow

https://github.com/Beraborrowofficial/blockend/pull/241 allowbreak #discussion_r2068501148

Issue H-3: getDebtToUnwindAndCollRequested() doesn't correctly handle collateral and debt calculation.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/30

Summary

`getDebtToUnwindAndCollRequested()` doesn't correctly handle collateral and debt calculation.

Vulnerability Detail

`getDebtToUnwindAndCollRequested()` function is used to calculate the amount of collateral and debt to withdraw from the Den, when users are executing withdraws from the vault.

The issue is this function currently doesn't correctly handle exposure tokens.

Assume a scenario where collateral (collVault) = 100, debt = 80, exposureToken = 120 (due to price increase), totalAssets() = 100 + 120 - 80 = 140.

When we withdraw all shares, the requestedAssets is calculated to 140 (by `_previewRedeem()` function) , and debtToUnwind is `80 * 140 / 100 = 112`). This means we will try to withdraw 140 collateral and repay 112 debt.

Ideally it should withdraw 100 collateral, and repay 80 debt.

```
function getDebtToUnwindAndCollRequested(uint256 _epoch) public view returns
↪  (uint256 debtToUnwind, uint256 collVaultSharesRequested, uint256 shareFee) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();
    ManagedLeveragedVaultStorage storage $ = _getManagedLeveragedVaultStorage();

    uint256 totalShares = $.reports[_epoch].totalShares;

    if (totalShares == 0) revert AmountZero();

    uint256 requestedAssets;
    (requestedAssets, shareFee) = _previewRedeem(totalShares);

    collVaultSharesRequested = _assetsToCollVaultShares(requestedAssets);

    (uint256 collVaultShares, uint256 debt) =
↪  IDenManager(b$.denManager).getDenCollAndDebt(address(this));

    // Find proportional debt unwind to collateral, so that ICR is unchanged, we
↪  assume an already pretty close ICR
```

```
    // If we were to responsabilize withdrawers to targetICR with a current
↪   deviated denICR, it would only responsabilize them to pay the slippage/borrow
↪   costs
    // Instead, we responsabilize both withdrawers and depositors, only using
↪   `increaseLeverage` and `decreaseLeverage`, which socializes losses
    debtToUnwind = debt * collVaultSharesRequested / collVaultShares; /// @dev
↪   Check `collVaultSharesRequested` can't be > `collVaultShares`
}
```

Impact

Withdraws aren't correctly handled.

Recommendation

Fix the withdraw logic by handling exposure tokens correctly.

Discussion

alex-beraborrow

`realizeProfits` handles this scenario

Issue H-4: realizeLoss() will always fail due to flashloan failure.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/38

Summary

`realizeLoss()` will always fail due to flashloan failure.

Vulnerability Detail

Note: this issue is found on the updated commit `8906ef180a2b0bb9fdb4e32d93e45819b4e540cb` where realizeProfit, realizeLoss features are applied.

`realizeLoss()` requires a flashloan from DebtToken. The DebtToken requires spending allowance from the flashloan receiver when repaying flashloan. However, in the the `onFlashLoan()` callback, there is no allowance approve, which means the flashloan will always fail.

https://github.com/Beraborrowofficial/blockend/blob/simplified_managed_leverage_vault/src/core/DebtToken.sol#L228

```
        ...
@>      _spendAllowance(address(receiver), address(this), amount + fee);
        _burn(address(receiver), amount);
```

Impact

`realizeLoss()` can never be called. This means when we have debtValue > exposureValue, we can't rebalance, and users may not be able to withdraw.

Recommendation

Approve allowance in `onFlashLoan()` function.

Discussion

alex-beraborrow

Agreed and fixed

Issue H-5: Users withdraw less asset when pnl is negative.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/39

Summary

Users withdraw less asset when pnl is negative.

Vulnerability Detail

Note: this issue is found on the updated commit `8906ef180a2b0bb9fdb4e32d93e45819b4e540cb` where realizeProfit, realizeLoss features are applied.

When executing a withdraw, we calculate the amount of collateral asset that should be given to the users. However, when pnl is negative (exposureValue < debtValue), users are withdrawing less amount then expected.

When we are calculating `collVaultSharesRequested`, we shouldn't deduct the `$.reports[_epoch].collLoss`, because the loss are already reflected in `totalAssets()`, and is applied in `_previewRedeem()` function.

For example, coll=100, debt=80, exposure=60 (due to price decrease). Assume we withdraw all of the shares, we need to call `realizeLoss()` first. Then we get coll=80, debt=60, exposure=60 (assume no slippage), with `$.reports[_epoch].collLoss` = 20. Then, when we execute the withdraw, the actual collVaultSharesRequested = 80-20 = 60, when it should actually be 80.

```
    function getDebtToUnwindAndCollRequested(uint256 _epoch) public view returns
↪   (uint256 debtToUnwind, uint256 collVaultSharesRequested, uint256 shareFee,
↪   uint256 currICR) {
        ...
        uint256 requestedAssets;
@>      (requestedAssets, shareFee) = _previewRedeem(totalShares);

        /// @dev Substract collLoss to responsabilize these withdrawers of their
↪   respective negative PnL
@>      collVaultSharesRequested = _assetsToCollVaultShares(requestedAssets) -
↪   $.reports[_epoch].collLoss;

        (uint256 collVaultShares, uint256 debt) =
↪   IDenManager(b$.denManager).getDenCollAndDebt(address(this));

        currICR = BeraborrowMath._computeCR(collVaultShares, debt,
↪   IDenManager(b$.denManager).fetchPrice());
```

```
        // Find proportional debt unwind to collateral, so that ICR is unchanged,
↪  we assume an already pretty close ICR
        // If we were to responsabilize withdrawers to targetICR with a current
↪  deviated denICR, it would only responsabilize them to pay the slippage/borrow
↪  costs
        // Instead, we responsabilize both withdrawers and depositors, only using
↪  `increaseLeverage` and `decreaseLeverage`, which socializes losses
        debtToUnwind = debt.mulDiv(collVaultSharesRequested, collVaultShares); ///
↪  @dev Check `collVaultSharesRequested` can't be > `collVaultShares`
    }
```

Impact

Loss of funds for users withdraw when pnl is negative.

Recommendation

Don't subtract the `$.reports[_epoch].collLoss`.

Discussion

GalloDaSballo

Agree that we should remove the code:
https://github.com/Beraborrowofficial/blockend/blob/43e303a0ff804a6e56f0665da6b9536b9faacd62/src/core/boyco/ManagedLeveragedVault.sol allowbreak #L814-L815

```
collVaultSharesRequested = _assetsToCollVaultShares(requestedAssets) -
↪  \$.reports[_epoch].collLoss;
```

And should exclusively get the vault shares, rounded down as to not cause another loss to the remaining depositors

Fix is WIP

alex-beraborrow

Think that just removing `collLoss` from everywhere will work

Issue M-1: ManagedLeveragedVault.sol::deposit() is missing slippage control

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/12

Summary

`ManagedLeveragedVault.sol::deposit()` doesn't have slippage control for the shares minted.

Vulnerability Detail

The deposit function is as follows:

```
function deposit(uint256 assets, address receiver, AddCollParams calldata params)
    public
    notPaused
    nonReentrant
    returns (uint256 shares)
{
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    shares = _deposit(assets, receiver);

    IERC20(asset()).forceApprove(address(b$.collVault), assets);
    uint256 collVaultShares = b$.collVault.deposit(assets, address(this));

    if (collVaultShares < params.minSharesOut) revert
↪   VaultSlippage(params.minSharesOut, collVaultShares);

    _addColl(collVaultShares, params.upperHint, params.lowerHint);
}
```

Note that there is no slippage control on the shares amount minted, whose ratio may change depending on several protocol actions or price updates. There is slippage control for the `collVaultShares`, but this may still mean a lower amount of shares as they are not 100% related.

Impact

User takes slippage losses.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR273

Tool Used

Manual Review

Recommendation

Add slippage control to shares.

Discussion

alex-beraborrow

Added on last commit hash update

Issue M-2: DoSed withdrawals due to Managed LeveragedVault::executeWithdrawalEpoch() repaying debt above limit

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/15

Summary

`ManagedLeveragedVault::executeWithdrawalEpoch()` repays debt to withdraw collateral from the Den to send to users, but it doesn't handle when the minimum debt limit is reached and becomes DoSed.

Vulnerability Detail

`ManagedLeveragedVault::executeWithdrawalEpoch()` is as follows:

```
function executeWithdrawalEpoch(
    ExecuteWithdrawalParams calldata params
) external onlyOwner nonReentrant claimCollateralSurplus(params.upperHint,
↪  params.lowerHint) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    _checkEpoch(params.epoch);

    CollDebt memory cd = _getCollVaultSharesAndDebtToUnwind(params.epoch);

    b$.borrowerOperations.adjustDen({
        denManager: b$.denManager,
        account: address(this),
        _maxFeePercentage: params.maxFeePercentage,
        _collDeposit: 0,
        _collWithdrawal: cd.collVaultSharesToWithdraw,
        _debtChange: cd.debtToUnwind,
        _isDebtIncrease: false,
        _upperHint: params.upperHint,
        _lowerHint: params.lowerHint
    });
   ...
}
```

If the chain of calls is followed, there is no limit to the `cd.debtToUnwind` calculated, and it may revert when everyone or many users try to withdraw in the same epoch (or last epochs), as the remaining debt would be lower than the min debt of the Den and it would

revert. In this case, it would be appropriate to close the Den and allow everyone to withdraw.

Impact

DoSed withdrawals.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR360

Tool Used

Manual Review

Recommendation

Close the Den if the remaining debt is below the minimum limit.

Discussion

alex-beraborrow

We don't deem necessary to close the den, we plan to have a protocol den through `ManagedLeveragedVault`:
https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/23allowbreak #issuecomment-2844744665

Issue M-3: ManagedLeveragedVault::execute WithdrawalEpoch() will never work because cd.prev ICR is not set

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/16

Summary

`ManagedLeveragedVault::executeWithdrawalEpoch()` asserts the new ICR is close to the old one, but the latter is never set. Marking medium as the contract is upgradeable.

Vulnerability Detail

The last check in `ManagedLeveragedVault::executeWithdrawalEpoch()` is:

```
_checkInvariantICR(getCurrentDenICR(), cd.prevICR, Tolerance.BOTH);
```

However, `cd.prevICR` is never set and it will always revert.

Impact

DoSed withdrawals

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR393

Tool Used

Manual Review

Recommendation

Set the old ICR.

Discussion

alex-beraborrow

Last commit update includes fix

Issue M-4: ManagedLeveragedVault::increase Leverage() fails when the debt is closed to the limit

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/18

Summary

`ManagedLeveragedVault::increaseLeverage()` will revert when the protocol is close to the max debt as it needs to borrow more than the debt allows in order for the ICR to reach the target.

Vulnerability Detail

`ManagedLeveragedVault::increaseLeverage()` check is too strict requiring that the final ICR is very close to the target:

```
_checkInvariantICR(getCurrentDenICR(), getTargetICR(), Tolerance.ABOVE);
```

Thus, it will not be possible to increase the leverage, even if this change would bring it closed to the target ICR.

Impact

Smaller leverage than intended.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR449

Tool Used

Manual Review

Recommendation

Allow the ICR to be higher than the target as long as it is above it.

Discussion

alex-beraborrow

Agreed, change (included in latest commit hash):

```
uint256 currentDenICR = getCurrentDenICR();
uint256 targetICR = getTargetICR();
if (currentDenICR < targetICR) revert PositionOutOfTargetCR(currentDenICR,
↪  targetICR);
```

Issue M-5: ManagedLeveragedVault::decrease Leverage() will not work when it goes below the minimum debt

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/20

Summary

`ManagedLeveragedVault::decreaseLeverage()` is used to increase the ICR of the Den, but it may be DoSed in case it goes below the minimum debt limit.

Vulnerability Detail

There is no consideration for the minimum debt in this function:

```
function decreaseLeverage(
    uint256 _exposureAmount,
    address _upperHint,
    address _lowerHint,
    ExternalRebalanceParams memory params
) external onlyOwnerOrKeeper nonReentrant returns (uint256 debt) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();
    ManagedLeveragedVaultStorage storage $ = _getManagedLeveragedVaultStorage();

    debt = _rebalance(
        RebalanceParams({
            sentCurrency: $.exposureToken,
            sentAmount: _exposureAmount,
            receivedCurrency: address(b$.nect),
            ext: params
        })
    );

    b$.borrowerOperations.repayDebt(
        b$.denManager,
        address(this),
        debt,
        _upperHint,
        _lowerHint
    );

    _checkInvariantICR(getCurrentDenICR(), getTargetICR(), Tolerance.ABOVE);
}
```

And the last check forces the ICR to reach the target. Hence, it may not be possible to deleverage at all risking liquidation. It should be possible to deleverage the maximum possible until the minimum debt is reached.

Impact

Liquidation/redemption risk.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR472

Tool Used

Manual Review

Recommendation

The target ICR requirement could be soften.

Discussion

alex-beraborrow

If we have to deleverage that much that we're close to `minNetDebt` (69 NECT) means that there is very few collateral left, meaning that impact is low. Prefer not to soften target ICR requirement to the downside since it can have worse side effects.

Also, that position that would be majoritarily lost is protocol owned.
https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/23 allowbreak #issuecomment-2844744665

Issue M-6: Protocol will need to donate minimum debt for all new ManagedLeveragedVaults

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/23

Summary

`ManagedLeveragedVault::deposit()` doesn't allow depositing before the Den is opened, which means the only way to have a minimum collateral amount to open the minimum debt in `ManagedLeveragedVault::openDen()` is by donating assets.

Vulnerability Detail

See `ManagedLeveragedVault::deposit()`:

```
function deposit(uint256 assets, address receiver, AddCollParams calldata params)
    public
    notPaused
    nonReentrant
    returns (uint256 shares)
{
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    shares = _deposit(assets, receiver);

    IERC20(asset()).forceApprove(address(b$.collVault), assets);
    uint256 collVaultShares = b$.collVault.deposit(assets, address(this));

    if (collVaultShares < params.minSharesOut) revert
    ↪  VaultSlippage(params.minSharesOut, collVaultShares);

    _addColl(collVaultShares, params.upperHint, params.lowerHint);
}
```

It tries to add collateral, but it reverts before the Den is opened. As opening a Den requires minimum debt (and thus minimum collateral), the only way to have assets when opening is by donating.

Impact

Protocol needs to donate min debt, and can't recover these funds.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR507

Tool Used

Manual Review

Recommendation

Let users deposit before the Den is opened (just let the funds idle instead of adding collateral).

Discussion

alex-beraborrow

Agree on the part the protocol can't withdraw its funds so that the last user withdrawals are not DOSed since there has to remain `minNetDebt`, but `openDen` already opinionate that it's the protocol the one supplying `minNetDebt`

Issue M-7: Missing several safeERC20 functions

Source:
https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/24

Summary

Some tokens will not work currently due to not using `safeERC20`.

Vulnerability Detail

`approve()`, `transferFrom()` don't work for all tokens and `safeERC20` should be used.

Impact

Protocol can't be used for certain tokens. Leaving as medium because vaults will be migrated and may have these tokens already with TVL.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR523
https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR493
https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR939

Tool Used

Manual Review

Recommendation

Use the `SafeERC20` lib.

Discussion

alex-beraborrow

All were added in last commit hash update.

Issue M-8: ManagedLeveragedVault::execute WithdrawalEpoch() incorrect ICR and DoSed withdrawals due to not accouting for fees

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/27

Summary

`ManagedLeveragedVault::executeWithdrawalEpoch()` applies the exposure -> nect swap fees loss on the users withdrawing, ending up repaying more debt than the collateral withdrawn, increasing the ICR and possibly reverting and DoSing withdrawals.

Vulnerability Detail

`ManagedLeveragedVault::executeWithdrawalEpoch()` repays and withdraws:

```
function executeWithdrawalEpoch(
    ExecuteWithdrawalParams calldata params
) external onlyOwner nonReentrant claimCollateralSurplus(params.upperHint,
↪  params.lowerHint) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    _checkEpoch(params.epoch);

    CollDebt memory cd = _getCollVaultSharesAndDebtToUnwind(params.epoch);

    b$.borrowerOperations.adjustDen({
        denManager: b$.denManager,
        account: address(this),
        _maxFeePercentage: params.maxFeePercentage,
        _collDeposit: 0,
        _collWithdrawal: cd.collVaultSharesToWithdraw,
        _debtChange: cd.debtToUnwind,
        _isDebtIncrease: false,
        _upperHint: params.upperHint,
        _lowerHint: params.lowerHint
    });
    ...
}
```

The math is supposed to keep the ICR just the same as before the withdrawal, which is attempted in `getDebtToUnwindAndCollRequested()`:

```
function getDebtToUnwindAndCollRequested(uint256 _epoch) public view returns
↪  (uint256 debtToUnwind, uint256 collVaultSharesRequested, uint256 shareFee) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();
    ManagedLeveragedVaultStorage storage $ = _getManagedLeveragedVaultStorage();

    uint256 totalShares = $.reports[_epoch].totalShares;

    if (totalShares == 0) revert AmountZero();

    uint256 requestedAssets;
    (requestedAssets, shareFee) = _previewRedeem(totalShares);

    collVaultSharesRequested = _assetsToCollVaultShares(requestedAssets);

    (uint256 collVaultShares, uint256 debt) =
↪  IDenManager(b$.denManager).getDenCollAndDebt(address(this));

    // Find proportional debt unwind to collateral, so that ICR is unchanged, we
↪  assume an already pretty close ICR
    // If we were to responsabilize withdrawers to targetICR with a current
↪  deviated denICR, it would only responsabilize them to pay the slippage/borrow
↪  costs
    // Instead, we responsabilize both withdrawers and depositors, only using
↪  `increaseLeverage` and `decreaseLeverage`, which socializes losses
    debtToUnwind = debt * collVaultSharesRequested / collVaultShares; /// @dev
↪  Check `collVaultSharesRequested` can't be > `collVaultShares`
}
```

However, after this debt is calculated to have the exact ICR ratio as before, the shares withdrawn will be further reduced by the slippage loss when getting the debt (NECT) from the exposure token:

```
function getCollVaultSharesToWithdraw(uint256 _collVaultSharesRequested, uint256
↪  _debtToUnwind, uint256 _exposureWithdrawn) public view returns (uint) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    address collVault = address(b$.collVault);

    // Instead of recalculating totalAssets, which socializes the loss to everyone
    // Let's see how much the loss was, and apply it to the withdrawer by
↪  substracting the collVaultShares to withdraw
    uint256 exposureWithdrawnValue = _getExposureValue(_exposureWithdrawn);
    /// @dev Most cases will present a loss, but in case of profit, we don't add
↪  coll
    uint256 slippageLossInUsd = exposureWithdrawnValue > _debtToUnwind ?
↪  exposureWithdrawnValue - _debtToUnwind : 0;
```

```
    uint256 slippageLossInColl =
↪   slippageLossInUsd.convertToColl(getPrice(collVault),
↪   IAsset(collVault).decimals(), Math.Rounding.Up);
    return _collVaultSharesRequested - slippageLossInColl;
}
```

Thus, the final ICR will be incorrect and the withdrawal may be DoSed due to this `_checkInvariantICR(getCurrentDenICR(), cd.prevICR, Tolerance.BOTH);`.

Impact

DoSed withdrawals.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR819

Tool Used

Manual Review

Recommendation

The debt to repay in `getDebtToUnwindAndCollRequested()` must taken into account the exit fees of the exposure token. This has a closed formula by using these 3 invariants:

1. Computing final totalAssets() to maintain the same ratio in the vault.
2. Debt_final = Collateral_final * ICR
3. Exposure_delta = Debt_delta / (1 - lspExitFees) Leading to:

Discussion

alex-beraborrow

Agreed, but won't fix since we'll whitelist the `ManagedLeveragedVault` to not pay withdrawal fees on neither the 2 planned exposure tokens (SNECT and BB.SNECT). Rendering the scenario in which there's `slippageLossInColl` unrealistic.

Issue M-9: openDen() may DoS due to borrowed debt lesser than minDebt.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/31

Summary

In `openDen()`, a new Den is created and borrows a minimum amount of debt to pass the minDebt check in DenManager. However, this may fail due to not enough borrowed debt.

Vulnerability Detail

See the check in BorrowerOperations.sol. In non recovery mode, a borrowingFee of `getBorrowingRateWithDecay()` is added to initial debt. It rounds down here.

```
if (!isRecoveryMode) {
    vars.netDebt = vars.netDebt + _triggerBorrowingFee(denManager, account,
↪   _maxFeePercentage, _debtAmount);
}
_requireAtLeastMinNetDebt(vars.netDebt);
```

In `openDen()` code, when calculating the borrowed debt `minNetDebt`, there are two issues:

1. Rounding. For example, if borrowFee = 0.1, the calculated debt rounds down here. A rough intuitive example is: `minNetDebt = 95, calculated debt = 95 / 1.1 = 86, actual net debt = 86 + 86 * 0.1 = 86 + 8 = 94 < 95`.
2. If Den is in recovery mode, there will be no borrow fees, so the calculated debt will always be shorter than `minNetDebt`.

```
// @audit-bug: This may DoS due to rounding.
uint256 minNetDebt = b$.borrowerOperations.minNetDebt() * WAD / (WAD +
↪   denManager.getBorrowingRateWithDecay());

b$.collVault.approve(address(b$.borrowerOperations), collVaultShares);
b$.borrowerOperations.openDen(
    address(denManager),
    address(this),
    _maxFeePercentage,
    collVaultShares,
    minNetDebt,
    _upperHint,
    _lowerHint
);
```

Impact

`openDen()` will Dos.

Recommendation

1. For non-recovery mode, round up the calculation.
2. For recovery mode, just use `borrowerOperations.minNetDebt()`.

Or for simplicity, always use `borrowerOperations.minNetDebt()`.

Discussion

alex-beraborrow

Rounding up:

```
uint256 minNetDebt = b\$.borrowerOperations.minNetDebt().mulDiv(WAD, (WAD +
↪  denManager.getBorrowingRateWithDecay()), Math.Rounding.Up);
```

Issue M-10: ManagedLeveragedVault will brick if Den is liquidated or fully redeemed.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/32

Summary

ManagedLeveragedVault will brick if Den is liquidated or fully redeemed.

Vulnerability Detail

If the Den was somehow liquidated/fully redeemed and closed. This vault will end up with a bunch of exposure token (either sNECT or bb.sNECT).

Currently there is no way to convert these token back to underlying asset token, and users can't withdraw their tokens back (because `executeWithdrawalEpoch()` function must interact with an open Den).

If we try reopen the Den again, there will be no collateral token to begin with.

Impact

ManagedLeveragedVault will brick if Den is liquidated or fully redeemed.

Recommendation

Add an external admin function to convert exposure token -> underlying asset, like what we did in BoycoVault.

Discussion

alex-beraborrow

`realizeProfits` handles this scenario

Issue M-11: _getCollVaultSharesAndDebtToUnwind() may not acquire enough NECT due to extra collaterals.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/36

Summary

`_getCollVaultSharesAndDebtToUnwind()` may not acquire enough NECT due to extra collaterals.

Vulnerability Detail

When we are conducting a withdraw from vault, we need to withdraw collateral and repay NECT debt into the Den. Since there is no NECT in the vault, we need to convert exposure tokens into NECT.

Exposure token can either be sNECT (LiquidStabilityPool) or bb.sNECT (CollateralVault of sNECT). Both contracts contain "extra tokens", and when calling `withdraw()`, we may not get enough sNECT/NECT tokens as expected.

```
uint256 exposureWithdrawn;
if ($.exposureToken != address(b$.lsp)) {
    uint256 lspNeeded = IERC4626(b$.lsp).previewWithdraw(cd.debtToUnwind);
    exposureWithdrawn = IERC4626($.exposureToken).withdraw(lspNeeded,
↪   address(this), address(this));
}
// LSP is asset of BB.SNECT
IERC4626(b$.lsp).withdraw(cd.debtToUnwind, address(this), address(this));
```

Take bb.sNECT as an example, when calling `withdraw()`, it may return reward tokens other than sNECT. For example, if there were 100 sNECT and 50 rewardToken in bb.sNECT (assume 1 sNECT = 1 rewardToken), when we call `withdraw(asset = 50)`, we would end up with 33.3333 sNECT and 16.666 rewardToken.

```
    function withdraw(
        uint assets,
        address receiver,
        address _owner
    ) public override(ERC4626Upgradeable, IERC4626) harvestRewards returns (uint
↪   shares) {
        ...
        _withdraw(msg.sender, receiver, _owner, assetAmount, shares);
```

```
@>      _withdrawExtraRewardedTokens(receiver, netShares, _totalSupply);
    }
```

The same issue applies for sNECT as well.

Impact

During withdraw, `_getCollVaultSharesAndDebtToUnwind()` will not acquire NECT for repay, and withdraw will fail.

Recommendation

Make sure we have enough NECT tokens for withdraw, either by rebalancing in sNECT/bb.sNECT to make sure there is only one token, or by adjusting the withdraw amount.

Discussion

alex-beraborrow

BB.SNECT is auto-compounding, so no more than 0.001% of it's composition should be rewards in any time. SNECT is also actively rebalanced to NECT if there are any liquidated collaterals by our own keepers.

Issue M-12: ICR check in onFlashLoan() callback may cause DoS.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/40

Summary

ICR check in `onFlashLoan()` callback may cause DoS.

Vulnerability Detail

Note: this issue is found on the updated commit `8906ef180a2b0bb9fdb4e32d93e45819b4e540cb` where realizeProfit, realizeLoss features are applied.

At the end of `onFlashLoan()` callback, there is a check that checks ICR does not deviate too large. However, considering how flashloan is being used, this may cause DoS.

Flashloans are used to repay debt in Den, then approximately the same value of collateral is withdrawn from Den, swap to NECT and repay flashloan. Flashloans are used when calling `realizeLoss()`.

For example, if coll=100, debt=80, exposure=60, ICR=100/80=1.25. When we realize loss for all shares, we withdraw 20 collateral and repay 20 debt, then coll=80, debt=60, exposure=60, ICR=80/60=1.333. In an extreme scenario if exposure=0, we will end up with coll=20, debt=0, with ICR = infinite large.

The ICR deviates a lot, and may fail on the invariant check.

```
_checkInvariantICR(getCurrentDenICR(), prevICR, Tolerance.BOTH);
```

Impact

`realizeLoss()` may fail. This means when we have debtValue > exposureValue, we can't rebalance, and users may not be able to withdraw.

Recommendation

Remove the check.

Discussion

alex-beraborrow

Agreed and fixed

Issue M-13: exposureWithdrawn is not correctly calculated during withdraws.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/41

Summary

`exposureWithdrawn` is not correctly calculated during withdraws.

Vulnerability Detail

Note: this issue is found on the updated commit `8906ef180a2b0bb9fdb4e32d93e45819b4e540cb` where realizeProfit, realizeLoss features are applied.

In `_getCollVaultSharesAndDebtToUnwind()` function, we convert exposure token to NECT to repay debt. There may be some slippage loss when the conversion is done, so we need to apply the slippage loss to the withdrawers.

`exposureWithdrawn` is the amount of exposure token being converted, and is used to calculate slippage loss. The bug here is, when exposure token is sNECT (LSP), it is always zero.

```
uint256 exposureWithdrawn;
if ($.exposureToken != address(b$.lsp)) {
    uint256 lspNeeded = IERC4626(b$.lsp).previewWithdraw(cd.debtToUnwind);
    exposureWithdrawn = IERC4626($.exposureToken).withdraw(lspNeeded,
↪   address(this), address(this));
}
// LSP is asset of BB.SNECT
IERC4626(b$.lsp).withdraw(cd.debtToUnwind, address(this), address(this));

cd.collVaultSharesToWithdraw =
↪   getCollVaultSharesToWithdraw(collVaultSharesRequested, cd.debtToUnwind,
↪   exposureWithdrawn);
```

Impact

If exposure token is sNECT, the slippage loss during withdraw will always be zero, and all users would be affected from this loss.

Recommendation

Calculate `exposureWithdrawn` correctly.

Discussion

alex-beraborrow

Already fixed on latest commit hash
https://github.com/Beraborrowofficial/blockend/pull/241
allowbreak #discussion_r2068501148

GalloDaSballo

IMO this needs to be fixed by checking the delta total assets That will be the loss that the operation has locked This loss should be fully imputed to the withdrawers

Issue L-1: Unused/useless code

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/11

Summary

There are multiple unused pieces of code.

Vulnerability Detail

See links below.

Impact

Deployment costs

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR135
https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR1089
https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR382

Tool Used

Manual Review

Recommendation

Remove this code.

Discussion

alex-beraborrow

Removed on last commit hash update

Issue L-2: ManagedLeveragedVault::deposit() slippage control on collVaultShares is not intuitive

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/13

Summary

`ManagedLeveragedVault::deposit()` has slippage control for `collVaultShares` but it applies to the full deposited collateral when the user only gets a share of it due to entry fees.

Vulnerability Detail

The deposit function is:

```
function deposit(uint256 assets, address receiver, AddCollParams calldata params)
    public
    notPaused
    nonReentrant
    returns (uint256 shares)
{
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    shares = _deposit(assets, receiver);

    IERC20(asset()).forceApprove(address(b$.collVault), assets);
    uint256 collVaultShares = b$.collVault.deposit(assets, address(this));

    if (collVaultShares < params.minSharesOut) revert
 ↪  VaultSlippage(params.minSharesOut, collVaultShares);

    _addColl(collVaultShares, params.upperHint, params.lowerHint);
}
```

The user deposits assets and pays entry fees. However, the `params.minSharesOut` parameter is compared against `collVaultShares`, which comes from converting the whole assets deposited into collateral vault shares. This can be unexpected for users as slippage control on a deposit function should be on the amount the user actually gets (assets minus entry fees), but it currently compares against all assets deposited.

Impact

UX issues. Some users may be caught off guard but it only leads to reverts at max.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR278

Tool Used

Manual Review

Recommendation

Apply slippage control to the net `collVaultShares` the user mints.

Discussion

alex-beraborrow

Added on last commit hash update

Issue L-3: Incorrect ERC4626ExceededMaxRedeem event on ManagedLeveragedVault.sol:: cancelWithdrawalIntent()

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/14

Summary

`ERC4626ExceededMaxRedeem` first argument is `owner`, but the code emits it with receiver.

Vulnerability Detail

Full function:

```
function cancelWithdrawalIntent(uint256 epoch, uint256 sharesToCancel, address
↪  receiver) external nonReentrant {
    /// @dev To be tested, but I think that between CUTOFF and CUTOFF - EPOCH (60
↪  mins and 45 mins) before end, it can be canceled
    if (epoch < getWithdrawalRequestEpoch(block.timestamp)) revert CancelTooLate();

    ManagedLeveragedVaultStorage storage $ = _getManagedLeveragedVaultStorage();

    if ($.reports[epoch].reported) revert AlreadyReported();
    uint256 shares = $.reports[epoch].balanceOf[msg.sender];
    if (shares == 0) revert AmountZero();
    if (sharesToCancel > shares) revert ERC4626ExceededMaxRedeem(msg.sender,
↪  sharesToCancel, shares);

    $.reports[epoch].balanceOf[msg.sender] -= sharesToCancel;
    $.reports[epoch].totalShares -= sharesToCancel;

    _transfer(address(this), receiver, sharesToCancel);

    emit WithdrawalIntentCanceled(msg.sender, receiver, epoch, sharesToCancel);
}
```

It shows that `ERC4626ExceededMaxRedeem` is emitted with `msg.sender`, which is the `receiver`, not the `owner`.

Impact

Incorrect event emission.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR331‘

Tool Used

Manual Review

Recommendation

Use another event.

Discussion

alex-beraborrow

Another custom error used, fix included in last commit hash

Issue L-4: ManagedLeveragedVault::increaseLeverage() will never work in Recovery mode

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/17

Summary

Recovery mode doesn't allow Dens to decrease their ICR. Hence, `ManagedLeveragedVault::increaseLeverage()` will revert when the protocol is in Recovery mode.

Vulnerability Detail

`ManagedLeveragedVault::increaseLeverage()` withdraws debt:

```
b$.borrowerOperations.withdrawDebt(
    b$.denManager,
    address(this),
    _maxFeePercentage,
    _debtAmount,
    _upperHint,
    _lowerHint
);
```

But this is not possible in Recovery mode, as it requires that the new ICR is bigger than the old one.

Impact

DoSed increasing leverage.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR431-R438

Tool Used

Manual Review

Recommendation

Nothing can be really done but it's something to keep in mind.

Discussion

alex-beraborrow

Agreed, important to monitor and advise in fe, shouldn't be a big problem since we plan to have `targetICR` within a great range above `CCR`

Issue L-5: ManagedLeveragedVault::getAvailable Debt() used in ManagedLeveragedVault::increase Leverage() is incorrect

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/19

Summary

`ManagedLeveragedVault::increaseLeverage()` allows borrowing until the `maxSystemDebt` limit, but the check doesn't account for the borrowing fee triggered.

Vulnerability Detail

This is the `ManagedLeveragedVault::getAvailableDebt()` function:

```
function getAvailableDebt() internal view returns (uint256) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    IDenManager denManager = IDenManager(b$.denManager);

    uint256 borrowCap = denManager.maxSystemDebt();
    uint256 totalActiveDebt = denManager.getTotalActiveDebt();
    uint256 defaultedDebt = denManager.defaultedDebt();

    return borrowCap - totalActiveDebt - defaultedDebt;
}
```

It doesn't account for the borrowing fee triggered, which means the max debt will be overestimated.

Impact

This function would be used most likely offchain to calculate the max debt to leverage up, which would revert.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR428

Tool Used

Manual Review

Recommendation

This function should return the debt multiplied by (1 - borrowingFee).

Discussion

alex-beraborrow

Removed `getAvailableDebt()` since it's not critical due to `denManager` already does the check.

Issue L-6: ManagedLeveragedVault::donate Collateral() is missing a slippage check

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/21

Summary

`ManagedLeveragedVault::donateCollateral()` doesn't control slippage, leading to losses

Vulnerability Detail

The function is:

```
function donateCollateral(uint256 _collateralAmount, address _upperHint, address
↪  _lowerHint) external onlyOwner {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    uint256 assetsNeeded = b$.collVault.previewMint(_collateralAmount);
    IERC20(asset()).transferFrom(msg.sender, address(this), assetsNeeded);

    IERC20(asset()).forceApprove(address(b$.collVault), assetsNeeded);
    b$.collVault.mint(_collateralAmount, address(this));

    IERC20(address(b$.collVault)).forceApprove(address(b$.borrowerOperations),
↪  _collateralAmount);
    _addColl(_collateralAmount, _upperHint, _lowerHint);

    // TODO consider invariants
}
```

As can be seen, there is no slippage check in the coll vault, possibly leading to fund loss. It may be mitigated by the approval amount.

Impact

Collateral vaults are not really manipulatable but still it may be possible for the price to change too much.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR492

Tool Used

Manual Review

Recommendation

Add a slippage check.

Discussion

alex-beraborrow

If a CollVault has rewardedTokens is rebalanced constantly to compound the asset. Also `donateCollateral` was removed due to bytecode size issues. It will be temporally added in an upgrade in the case of bad debt.

Issue L-7: Some view functions will not work under certain conditions

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/25

Summary

`ManagedLeveragedVault::getStrategyCR()`, `ManagedLeveragedVault::exposurePositionWeight()` and `ManagedLeveragedVault:: getStrategyLeverage()` may revert.

Vulnerability Detail

The functions below when there is no debt or position, most likely when the Den hasn't been opened or when the Vault was just deployed.

```
function getStrategyCR() public view returns (uint256) {
    return (getMarginValue() + getExposureBalance()) * WAD / getDebtBalance();
}

function exposurePositionWeight() public view returns (uint256) {
    return getExposureValue() * WAD / getCurrentPositionValue();
}

function collateralPositionWeight() public view returns (uint256) {
    return WAD - exposurePositionWeight();
}

function getStrategyLeverage() public view returns (uint256) {
    uint256 positionValue = getCurrentPositionValue();
    return positionValue * WAD / (positionValue - getDebtBalance());
}
```

Impact

View function reverts.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR742
https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR746

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR753

Tool Used

Manual Review

Recommendation

Handle these cases.

Discussion

alex-beraborrow

Functions were removed due to unuse and to reduce bytecode size limits

Issue L-8: ManagedLeveragedVault::getDebtToUnwindAndCollRequested() is inaccurate when there are surplus tokens

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/26

Summary

`ManagedLeveragedVault::getDebtToUnwindAndCollRequested()` is called on chain after the modifier `claimCollateralSurplus` is called, so it is updated, but offchain this modifier was not called and the view function is off.

Vulnerability Detail

`ManagedLeveragedVault::getDebtToUnwindAndCollRequested()` returns the debt to unwind and coll requested, but doesn't include the coll surplus as a view function offchain:

```
function getDebtToUnwindAndCollRequested(uint256 _epoch) public view returns
↪  (uint256 debtToUnwind, uint256 collVaultSharesRequested, uint256 shareFee) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();
    ManagedLeveragedVaultStorage storage $ = _getManagedLeveragedVaultStorage();

    uint256 totalShares = $.reports[_epoch].totalShares;

    if (totalShares == 0) revert AmountZero();

    uint256 requestedAssets;
    (requestedAssets, shareFee) = _previewRedeem(totalShares);

    collVaultSharesRequested = _assetsToCollVaultShares(requestedAssets);

    (uint256 collVaultShares, uint256 debt) =
↪  IDenManager(b$.denManager).getDenCollAndDebt(address(this));

    // Find proportional debt unwind to collateral, so that ICR is unchanged, we
↪  assume an already pretty close ICR
    // If we were to responsabilize withdrawers to targetICR with a current
↪  deviated denICR, it would only responsabilize them to pay the slippage/borrow
↪  costs
    // Instead, we responsabilize both withdrawers and depositors, only using
↪  `increaseLeverage` and `decreaseLeverage`, which socializes losses
    debtToUnwind = debt * collVaultSharesRequested / collVaultShares; /// @dev
↪  Check `collVaultSharesRequested` can't be > `collVaultShares`
}
```

Essentially when it is called offchain `collVaultShares` doesn't include the surplus balance, but on chain it will always include it.

Impact

View function is incorrect.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR787

Tool Used

Manual Review

Recommendation

Include the surplus balance if it exists.

Discussion

alex-beraborrow

Agree, event if the contract the getter is used after claiming `collSurplus`, we'll fix this for offchain correctness:

```
(uint256 collVaultShares, uint256 debt) =
↪  IDenManager(b\$.denManager).getDenCollAndDebt(address(this));
collVaultShares += IDenManager(b\$.denManager).surplusBalances(address(this));
```

Issue L-9: Most ManagedLeveragedVault functions are DoSed due to bad debt check

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/28

Summary

Most `ManagedLeveragedVault` functions rely on `totalAssets()`, including ERC4626 compliant functions that must not revert; however, they will revert when there is bad debt.

Vulnerability Detail

`ManagedLeveragedVault:: _assetsValuation()` reverts when there is bad debt:

```
if (debtValue > collateralValue + exposureValue) {
    revert BadDebt(debtValue - (collateralValue + exposureValue));
}
```

This will make all ERC4626 functions revert that must not do so when called, becoming non compliant. Additionally, it is not possible to request withdrawal intents temporarily.

Impact

DoSed request intent and non ERC4626 compliance

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/pull/1/files#diff-2d972f52027e0ba1065b2de2b424416244bfef9e2a79d2604e1687867f72c91eR985

Tool Used

Manual Review

Recommendation

Handle the situation instead of reverting to allow the normal behaviour of the mentioned functions (although the behaviour may be hard to define).

Discussion

alex-beraborrow

When there's BadDebt, `totalAssets` is negative. The only way to get out of it is by the collateral value going up at some moment, or collateral being donated. We do want most functionality to revert until the above happens.

GalloDaSballo

I agree with the decision to not fix Also don't believe the vault will be ERC4626 compatible due to how it's built

alex-beraborrow

Agree, it's reported on the function natspec

Issue L-10: totalAssets check in initialize function doesn't work when Den is closed during migration.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/33

Summary

`totalAssets` check in initialize function doesn't work when Den is closed during migration.

Vulnerability Detail

In the initialize function, there is a check that the `totalAssets()` function should be within a expected range.

```
uint256 _totalAssets = totalAssets();
uint256 maxDelta = Math.min(params._previousTotalAssets, _totalAssets) / BP;
if (!BeraborrowMath._isApproxEqAbs(
    params._previousTotalAssets,
    _totalAssets,
    maxDelta
)) revert TotalAssetsDeviation();
```

However, if the Den is closed during migration, all assets would be in the form of underlying assets lying in contract balance, which is not picked up in `totalAssets()` function. The check won't work in this case.

```
/**
 * @notice Denominated in `asset()`, tracks collVaultShares (coll and margin) and
↪  substracts PnL of the exposure token, minus den debt
 * @dev Breaks ERC4626 since it can revert on BadDebt
 */
function totalAssets() public view override(IManagedLeveragedVault,
↪  ERC4626Upgradeable) returns (uint256) {
    uint256 collVaultShares = getCollVaultBalance();
    uint256 debt = getDebtBalance();
    uint256 exposure = getExposureBalance();

    return _assetsValuation(collVaultShares, debt, exposure);
}
```

Impact

`totalAssets` check in initializer during migration does not work if Den is closed.

Recommendation

Handle the case where Den is closed during migration by checking the amount of asset balance is within expected range instead.

Discussion

alex-beraborrow

Acknowledged

Issue L-11: ManagedLeveragedVault is not initialized correctly when deploying as a new contract.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/34

Summary

`ManagedLeveragedVault` is not initialized correctly when deploying as a new contract.

Vulnerability Detail

In the `initialize()` function, the current code only fills in the data in `ManagedLeveragedVaultStorage` storage, but doesn't initialize any value in `BoycoVaultStorage` or ERC4626/ERC20 storage.

This wouldn't be a problem if `ManagedLeveragedVault` is always updated from an old BoycoVault, but it would be an issue if `ManagedLeveragedVault` is deployed as a new contract.

Impact

`ManagedLeveragedVault` wouldn't work when deployed as a new contract.

Recommendation

Always make sure `ManagedLeveragedVault` is migrated from an old BoycoVault.

Discussion

alex-beraborrow

Already acknowledged in initial contract natspec.

Issue L-12: getStrategyCR() incorrectly adds up exposure token value twice.

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/35

Summary

`getStrategyCR()` incorrectly adds up exposure token value twice.

Vulnerability Detail

The CR for strategy should be `(collateral valut + exposure value - debt value) / debt value`, however exposure is added twice here.

```
    function totalAssets() public view override returns (uint256) {
        uint256 collVaultShares = getCollVaultBalance();
        uint256 debt = getDebtBalance();
        uint256 exposure = getExposureBalance();

        return _assetsValuation(collVaultShares, debt, exposure);
    }

    function getMarginValue() public view returns (uint256) {
        uint256 assetPrice = getPrice(asset());
        uint256 assets = totalAssets();
        uint8 assetDecimals = IAsset(asset()).decimals();

        return assets.convertToValue(assetPrice, assetDecimals);
    }

@>  function getStrategyCR() public view returns (uint256) {
        return (getMarginValue() + getExposureBalance()) * WAD / getDebtBalance();
    }
```

Impact

`getStrategyCR()` isn't used anywhere onchain, only impacts offchain or integrations.

Recommendation

Should be `return getMarginValue() * WAD / getDebtBalance();`

Discussion

alex-beraborrow

Agreed, removed due to bytecode size issues and no contract use.

Issue L-13: Hints on withdrawal may fail as the ICR changes between claiming collateral surplus and repaying debt

Source: https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/issues/37

Summary

Insertion hints depend on the ICR of the Den, on withdrawals the same hints are used in 2 different instances, which may not work and revert.

Vulnerability Detail

The `ManagedLeveragedVault::executeWithdrawalEpoch()` is as follows:

```
function executeWithdrawalEpoch(
    ExecuteWithdrawalParams calldata params
) external onlyOwner nonReentrant claimCollateralSurplus(params.upperHint,
↪  params.lowerHint) {
    BoycoVaultStorage storage b$ = _getBoycoVaultStorage();

    _checkEpoch(params.epoch);

    CollDebt memory cd = _getCollVaultSharesAndDebtToUnwind(params.epoch);

    b$.borrowerOperations.adjustDen({
        denManager: b$.denManager,
        account: address(this),
        _maxFeePercentage: 0,
        _collDeposit: 0,
        _collWithdrawal: cd.collVaultSharesToWithdraw,
        _debtChange: cd.debtToUnwind,
        _isDebtIncrease: false,
        _upperHint: params.upperHint,
        _lowerHint: params.lowerHint
    });
```

It uses the same hints for 2 different intances, with different ICRs each time. Hence, there is a chance the transaction reverts.

Impact

DoSed `ManagedLeveragedVault::executeWithdrawalEpoch()`. Workaround is possible by manually claiming collateral surplus before calling it.

Code Snippet

https://github.com/sherlock-audit/2025-04-beraborrow-vault-update/blob/main/blockend/src/core/boyco/ManagedLeveragedVault.sol#L297-L314

Tool Used

Manual Review

Recommendation

Send different hints.

Discussion

alex-beraborrow

Agree, if there is `collSurplus` added there will be a big deviation in NICR. We'll need to have our keeper looking at claimCollateral non-stop so that it's not batched into any other function in practicality

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.