Twyne - dynamic liquidation incentives

Prepared by: Josselin Feist (josselin@seceureka.com)

December 31th, 2025

Prepared for: ***Twyne***

Executive Summary

Overview

From December 15th to December 17th, we reviewed the dynamic liquidation incentives from PR187.

Two low severity findings and one informational finding were found. None of them would have had a significant impact on the protocol.

This document was part of a larger review that included the review of the health check removal, the dynamic liquidation incentive, and the aave's integration. This document contains only the dynamic liquidation incentives related findings.

After the fix review, the PR was merged into main (commit 88963747d7fb898b6138d8cd206073630013ba13)

Security findings

ID	Title	Severity	Likelihood
3	The dust in liquidator reward can be lost	*Low*	*Low*
4	Lack of slippage on liquidate and handleExternalLiquidation	*Low*	*Low*
5	Whitepaper inaccuracies	*Informational*	*Low*

The dust in liquidator reward can be lost			
Severity: Low	*Likelihood: Low*	*Status: fixed*	*003*

Summary

In case of a vault without debt after an external liquidation, the liquidator reward might be non-zero due to precision loss, while not being sent to the liquidator.

PR affected: PR187

Detailed description

During the external liquidation, if the borrowing amount left is zero, the liquidator reward should be zero as well. However, due to the rounding operation happening through the intermediate steps, the reward might be above zero.

The reward to be sent goes through multiple intermediate steps that include oracle's quote and share conversion:

```
        // Calculate userCollateral in USD
        uint C_new = twyneVaultManager.oracleRouter().getQuote(
                userCollateral,
                __asset,
                IEVault(__asset).unitOfAccount()
            );

        // borrow amount in USD
        (, uint B) = IEVault(targetVault).accountLiquidity(address(this),
true);

        // borrower's claim in `userCollateral` amount of collateral
        uint borrowerClaim = collateralForBorrower(B, C_new);
        uint liquidatorReward = userCollateral - borrowerClaim;
```

Figure 1: EulerCollateralVault.sol#L229-L241

```
    function _convertBaseToCollateral(uint collateralValue) internal view
virtual override returns (uint collateralAmount) {
        collateralAmount = twyneVaultManager.oracleRouter().getQuote(
```

```
                collateralValue,
                IEVault(intermediateVault).unitOfAccount(),
                IEVault(asset()).asset()
            );
        return Math.min(totalAssetsDepositedOrReserved - maxRelease(),
IEVault(asset()).convertToShares(collateralAmount));
    }
```

Figure 2: EulerCollateralVault.sol#L185-L192

In particular, when B is zero:

- `collateralForBorrower(B, C_new)` should return the user collateral
- `C_new` depends on the oracle's quote on the user collateral (Figure 1)
- Which is then converted back to user collateral value through another oracle's quote and a share conversion (Figure 2)

The oracle's back-and-forth and the share conversion can lead to a loss of precision. As a result, `collateralForBorrower(B, C_new)` might return a value slightly lower than the user collateral.

`handleExternalLiquidation` does not send the liquidator reward if `_maxRepay` is zero (i.e. B is zero):

```
        (uint liquidatorReward, uint releaseAmount, uint borrowerClaim) =
splitCollateralAfterExtLiq(amount, _maxRepay, _maxRelease);

        if (_maxRepay > 0) {
            // step 1: repay all external debt
            SafeERC20Lib.safeTransferFrom(IERC20_Euler(targetAsset),
liquidator, address(this), _maxRepay, permit2);
            IEVault(targetVault).repay(_maxRepay, address(this));

            // step 2: transfer collateral reward to liquidator
            SafeERC20Lib.safeTransfer(IERC20_Euler(__asset), liquidator,
liquidatorReward);
        }
```

Figure 3: EulerCollateralVault.sol#L247-L277

As a result, the reward will be lost if `_maxRepay` is zero.

Recommendations

Move the liquidator reward transfer outside of the `_maxRepay > 0` branch and send the reward if it is above zero in `handleExternalLiquidation`.

Consider also creating a short circuit-path if `_maxRepay` is zero to avoid all the intermediate steps, in particular if this path is expected to be common during liquidation.

Fix status

Fixed with PR PR220. A short circuit path was added if `_maxRepay` is zero, as result there is no dust left in this case.

Lack of slippage on liquidate and handleExternalLiquidation		
Severity: Low	*Likelihood: Low*	*004*

Summary

The lack of slippage on the `liquidate`/`handleExternalLiquidation` functions might lead the liquidator to earn less profit than anticipated

PR affected: PR187

Detailed description

When a liquidation happens, the liquidator sends to the borrower its share:

```
   function liquidate() external callThroughEVC nonReentrant {
       [..]

       (uint B, uint C) = _getBC();
       SafeERC20Lib.safeTransferFrom(IERC20_Euler(asset()), liquidator,
borrower, collateralForBorrower(B, C), permit2);
```

Figure 1: CollateralVaultBase.sol#L440-L450

Due to the market conditions (ex: the price of the collateral change), the liquidator might have to forward more collateral than anticipated to the borrower and receive less reward than expected.

A similar situation can happen on `handleExternalLiquidation`, where the borrower might receive less reward than initially expected.

Recommendations

Consider adding slippage parameters to `liquidate` and `handleExternalLiquidation`.

Whitepaper inaccuracies		
Severity: Informational	*Likelihood: Low*	*005*

Summary

The following contains inaccuracies in the whitepaper:

- The following sentence should use λt and not λt˜

> We do so by setting the liquidation incentive to zero whenever Borrowers are not reserving CLP funds. This occurs whenever $\tilde{\lambda}_t \leq \beta_{safe} \cdot \tilde{\lambda}_e$.

- Equation 40 should better define the boundaries of the inequalities:
  - If λt == βsafe · λ˜, it is unclear if the first or second equation line should be taken
  - Similary, if λt == λ˜max
  - The implementation currently uses strict inequality for the second line
  - *Due to the interpolation property of f(), the inaccuracies should not have an impact, provided f() is correctly chosen*

$$i_t^{liq}(\lambda_t) = \begin{cases} 0 & \lambda_t \leq \beta_{safe} \cdot \tilde{\lambda}_e \\ (1 - \tilde{\lambda}_t^{max}) \cdot f\left(\frac{\lambda_t - \beta_{safe} \cdot \tilde{\lambda}_e}{\tilde{\lambda}_t^{max} - \beta_{safe} \cdot \tilde{\lambda}_e}\right) & \beta_{safe} \cdot \tilde{\lambda}_e \leq \lambda_t \leq \tilde{\lambda}_t^{max} \\ 1 - \lambda_t & \lambda_t \geq \tilde{\lambda}_t^{max} \end{cases} \quad (40)$$

Recommendations

Fix the inaccuracies in the whitepaper

Code quality

The following recommendations aim to improve code quality and align the implementation with software development best practices:

- **Use `CollateralVaultBase` instead of `EulerCollateralVault` in `HealthStatViewer.health` when converting the interfaces.** The code accepts arbitrary collateral vaults, and not just Euler collateral vaults
- **Add whitepaper references to `CollateralVaultBase.collateralForBorrower`.** The function implements a variant of equation (40), where the branches follow a different order, and the result is the inverse of (40) (as it returns the collateral for borrowing instead of the liquidator reward)