Twyne - Aave integration

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

December 31th, 2025

Prepared for: **Twyne**

Executive Summary

Overview

From December 18th to December 24th, we reviewed the aave's integration, working from aave-prod (commit *bde552127b1011a74ec6da927ae9b7f168487b93*).

We focused on AaveV3ATokenWrapper and AaveV3CollateralVault and related components. The review looked for ways to steal funds from external users and from users interacting with the protocol (e.g., wrapper LP providers and collateral vault's owner). We focused on the interaction between the wrapper, the collateral vault, the intermediate vault, and Aave, and also reviewed each component in isolation. The review included checks on rounding and other math-related edge cases, as well as risks that could trap the system (e.g., during liquidation, through forced reverting conditions, and edge cases in external calls). The integration of Aave's v3.6 was also reviewed.

We relied on the whitepaper for formula-level correctness, and reviewed the wrapper transfer math against the provided notes. Coverage was more limited on decimals handling and on waToken/aToken/underlying sync assumptions (e.g., `_totalAssetsDepositedOrReserved` vs aToken scaled balance). The oracle integration and deployment scripts received a lighter pass.

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 aave's integration related findings.

Security findings

ID	Title	Severity	Likelihood
6	Risk if no liquidity on intermediate vault	*Informational*	*Low*
7	Risky boolean condition in _canLiquidate	*Informational*	*Low*
8	Interfaces mismatch	*Informational*	*Low*

Risk if no liquidity on intermediate vault		
Severity: Informational	*Likelihood: Low*	*006*

Summary

Some of the vault's operations will be blocked if the intermediate vault has not enough liquidity.

Branch reviewed: aave-prod.

Detailed description

In most of the operation, `_handleExcessCredit` is called, and might try to borrow additional collateral from the intermediate vault:

```
   function _handleExcessCredit(uint __invariantCollateralAmount)
internal override {
       uint vaultAssets = totalAssetsDepositedOrReserved;
       if (vaultAssets > __invariantCollateralAmount) {
           vaultAssets -= intermediateVault.repay(vaultAssets -
__invariantCollateralAmount, address(this));
       } else if (vaultAssets < __invariantCollateralAmount) {
           vaultAssets +=
intermediateVault.borrow(__invariantCollateralAmount - vaultAssets,
address(this));
       }

       IAaveV3ATokenWrapper(asset()).rebalanceATokens_CV(vaultAssets);
       totalAssetsDepositedOrReserved = vaultAssets;
   }
```

The function will revert if the intermediate vault does not have enough collateral. As a result, all the related operations will not be possible.

A user in a risky position might be prevented from depositing more collateral.

During the review, Twyne showed the plan for a user-level mitigation, allowing a user to unblock their vault if the intermediate vault does not have enough liquidity, by combining multiple batch operations (e.g by repaying part of the collateral directly through the vault).

Recommendations

Document the planned mitigation.

Risky boolean condition in `_canLiquidate`			
Severity: Informational	*Likelihood: Low*	*Status: Fixed*	*007*

Summary

`_canLiquidate` has an early boolean exit condition that prevents the check of all the liquidation conditions

Branch reviewed: aave-prod.

Detailed description

`_canLiquidate` is meant to check if the vault is liquidatable by

- The external protocol
- The twyne condition

```
   function _canLiquidate() internal view virtual override returns
(bool) {
       IAaveV3ATokenWrapper __asset = IAaveV3ATokenWrapper(asset());
       (, uint totalDebtBase,,,,uint hf) =
IAaveV3Pool(targetVault).getUserAccountData(address(this));

       // hf can go up to type(uint).max. This condition is to handle
overflow reverts
       // in the next condition
       if (hf > type(uint).max / MAXFACTOR){
           return false;
       }

       if (uint(twyneVaultManager.externalLiqBuffers(address(__asset)))
* hf < 1e18 * MAXFACTOR) {
           return true;
       }

       uint userCollateralValue =
           (totalAssetsDepositedOrReserved - maxRelease()) *
uint(__asset.latestAnswer()) / tenPowAssetDecimals;
       return (totalDebtBase * MAXFACTOR > twyneLiqLTV *
userCollateralValue);
```

```
    }
```

Figure 1: AaveV3CollateralVault.sol#L178-L189

However, the overflow protection (`hf > type(uint).max / MAXFACTOR`) creates an early exit, bypassing the twyne liquidation condition check.

As a result, if a vault were to be very healthy (`hf > type(uint).max / MAXFACTOR`) on the lending protocol, but liquidable based on twyne's condition, it would still be considered as non-liquidable.

We classified this issue as informational, since the current state of the checks, such a scenario cannot happen.

Recommendations

Integrate the overflow check into this branch:

```
      if (uint(twyneVaultManager.externalLiqBuffers(address(__asset))) *
hf < 1e18 * MAXFACTOR)
```

Consider documenting that `_canLiquidate` must be equivalent to "condition1 OR condition2".

Fix status

Fixed with 55590e89. The overflow check was integrated into the branching condition for the external protocol liquidation.

Interfaces mismatch			
Severity: Informational	*Likelihood: Low*	*Status: Partially fixed*	*008*

Summary

Several solidity interfaces used throughout the Aave's integration are misleading

Branch reviewed: aave-prod.

Detailed description

For example:

- `IEVault.redeem` is used in `redeemUnderlying` on asset(), which is a `IAaveV3ATokenWrapper`
    - Note: both have a redeem function with the same interface
- `IEVault.decimals()` is used for the asset, instead of ERC20 in `AaveV3CollateralVault.initialize`
- `IERC20` is renamed to `IERC20_Euler` in `AaveV3CollateralVault` and in `CollateralVaultBase` (while they are used in non-euler setup)

Recommendations

Use the correct interfaces.

Consider adding interface checks during the internal code review process

Fix status

Partially fixed with 55590e89. Some interfarces were changed (i.e. `IEVault.decimals()`). However IERC20_Euler and `IEVault.redeem` are still used.

While they don't prevent the correct behavior of the contracts as the interfaces are compatible, their naming is still increasing the code complexity.

Code quality

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

- **Use POOL.supply instead of POOL.deposit for Aave's pool.** Deposit is deprecated: **Pool.sol#L782**