yAudit

January 15, 2026

Prepared for
Twyne

Audited by
HHK
adriro

Twyne Aave Integration

Smart Contract Security Assessment

Contents

1 Review Summary

1.1 Protocol Overview

Twyne is a credit delegation protocol that lets borrowers rent unused borrowing power from other lenders to boost their Liquidation LTV. Lenders earn additional yield while borrowers get to ramp up their leverage or insulate their debt.

1.2 Audit Scope

This audit covers 9 smart contracts totaling approximately 1150 lines of code across 3.5 days of review.

```
0xTwyne/twyne-contracts/src
├── Periphery
│   └── AaveV3Wrapper.sol
├── TwyneFactory
│   └── CollateralVaultFactory.sol
└── twyne
    ├── AaveV3ATokenWrapperOracle.sol
    ├── AaveV3CollateralVault.sol
    ├── CollateralVaultBase.sol
    ├── EulerCollateralVault.sol
    └── VaultManager.sol

0xTwyne/aave-v3-aToken-wrapper/src
├── AaveV3ATokenWrapper.sol
└── CustomERC4626StataTokenUpgradeable.sol
```

1.3 Risk Assessment Framework

1.3.1 Severity Classification

Severity	Description	Potential Impact
Critical	Immediate threat to user funds or protocol integrity	Direct loss of funds, protocol compromise
High	Significant security risk requiring urgent attention	Potential fund loss, major functionality disruption
Medium	Important issue that should be addressed	Limited fund risk, functionality concerns
Low	Minor issue with minimal impact	Best practice violations, minor inefficiencies
Undetermined	Findings whose impact could not be fully assessed within the time constraints of the engagement. These issues may range from low to critical severity, and although their exact consequences remain uncertain, they present a sufficient potential risk to warrant attention and remediation.	Varies based on actual severity
Gas	Findings that can improve the gas efficiency of the contracts.	Increased transaction costs
Informational	Code quality and best practice recommendations	Reduced maintainability and readability

Table 1: severity classification

1.4 Key Findings

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	2
Low	3
Informational	7

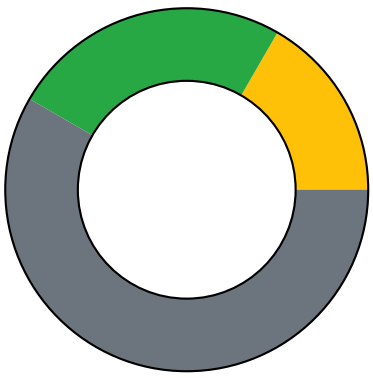

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

The audit of Twyne's Aave V3 integration identified no critical or high-severity vulnerabilities, indicating a solid implementation that maintains the security standards of the protocol's foundation. Two medium-severity findings were discovered, primarily related to operational issues under high utilization scenarios and incompatibilities between Aave's reward mechanism and Twyne's multi-vault architecture.

2 Audit Overview

2.1 Project Information

Protocol Name: Twyne
Repositories:

- https://github.com/0xTwyne/twyne-contracts
- https://github.com/0xTwyne/aave-v3-aToken-wrapper

Commit URLs:

- d83d60155f64a1f0fcd98a813038f6b37d2dd101
- ae1ca5641f7c148f97084640017cf6140a899183

2.2 Audit Team

HHK, adriro

2.3 Audit Timeline

The audit was conducted from November 3 to 6, 2025.

2.4 Audit Resources

Code repositories and documentation

2.5 Critical Findings

None.

2.6 High Findings

None.

2.7 Medium Findings

2.7.1 Redemption may fail due to an unavailable aToken backing

Technical Details

In `redeemUnderlying()` and `handleExternalLiquidation()`, the functions attempt to redeem wrapped aTokens for underlying assets and transfer them to the user. However, during periods of high utilization of the intermediate pool, most aTokens may be transferred to collateral vaults and are unavailable on the wrapped contract.

Both functions call `redeem()` before `rebalanceATokens_CV()`, which requires the wrapped tokens to be redeemed without first retrieving aTokens from the collateral vault. During high intermediate vault utilization, this causes redemption to fail and the transaction to revert, even though the tokens are technically available on the collateral vault.

Impact

Medium. `redeemUnderlying()` and `handleExternalLiquidation()` may revert when intermediate vault utilization is high.

Recommendation

Call `rebalanceATokens_CV()` before `redeem()` in `handleExternalLiquidation()`. Override `redeemUnderlying()` in the AAVE integration vault and call `_handleExcessCredit()` before `redeem()`.

Developer Response

Fixed in PR#189.

2.7.2 Broken reward accounting in wrapped aToken contract

The wrapped aToken contract's reward accounting system is incompatible with the protocol's architecture, leading to lost farming rewards and incorrect reward distribution.

Technical Details

When users borrow wrapped aTokens from the intermediate vault, `rebalanceATokens_CV()` unwraps them on the collateral vault so AAVE recognizes the collateral. This leaves the wrapped aToken contract virtually backed rather than physically backed, as the underlying aTokens are held by the collateral vault.
The forked StataToken reward accounting assumes 100% of aTokens remain in the wrapped contract. However, aTokens used for borrowing are held by the collateral vault, which receives their farming rewards.
The contract `ERC20AaveLMUpgradeable` that is inherited by the wrapped aToken uses the `INCENTIVES_CONTROLLER`'s `getAssetIndex()` function to determine rewards. When looking at the `INCENTIVES_CONTROLLER` function code, we can observe that this value is determined using the total value of the aToken.

```
_getAssetIndex(
        rewardData,
        IScaledBalanceToken(asset).scaledTotalSupply(),
        10 ** _assets[asset].decimals
      );
```

This creates several issues:

- The wrapped contract incorrectly tracks rewards for aTokens it no longer holds
- Collateral vaults receive farming rewards but cannot claim them

- Users claiming rewards from the wrapped contract receive inflated amounts on a first-come, first-served basis
- Wrapped aTokens lent on the intermediate vault generate rewards that the vault cannot claim

Impact

Medium. The farming rewards system is broken, resulting in lost rewards and incorrect distributions.

Recommendation

Remove the current farming reward system from the wrapped aToken contract.
Instead, implement an `onlyOwner()` function allowing the Twyne admin to claim farming rewards from both the wrapped contract and collateral vaults. Consider redistributing rewards through a simpler system, such as a Merkle tree, that is compatible with intermediate vaults.

Developer Response

Fixed in PR#2 & PR#194.

2.8 Low Findings

2.8.1 Pausing does not affect `rebalanceATokens_CV()`

Technical Details

In the AaveV3ATokenWrapper contract, the `rebalanceATokens_CV()` is not affected by the emergency pause since it doesn't rely on `_update()`.

Impact

Low.

Recommendation

Add the `whenNotPaused` modifier to enable pauses on this functionality.

Developer Response

Fixed in PR#2.

2.8.2 Stale liquidation threshold if E-mode gets disabled

Technical Details

When querying the external liquidation LTV, the implementation of `_getAaveLiqLTV()` uses the corresponding E-mode liquidation threshold if a category is configured in the vault.

```
102:     function _getAaveLiqLTV() internal view returns (uint) {
103:         if (categoryId == 0) {
104:             (,,uint currentLiquidationThreshold,,,,,,) = aaveDataProvider.
getReserveConfigurationData(underlyingAsset);
105:             return currentLiquidationThreshold;
106:         } else {
107:             return IAaveV3Pool(targetVault).getEModeCategoryCollateralConfig(
categoryId).liquidationThreshold;
108:         }
109:     }
```

However, Aave internally checks if the selected E-mode is actually enabled for the asset. The function `calculateUserAccountData()` checks if the collateral is associated with the borrower's E-mode to apply the category configuration.

```
121:             vars.isInEModeCategory =
122:               params.userEModeCategory != 0 &&
123:               EModeConfiguration.isReserveEnabledOnBitmap(vars.
eModeCollateralBitmap, vars.i);
```

While it is expected that Twyne configures categories correctly, these settings can be modified at Aave after a vault has been created.

Impact

Low. The liquidation threshold might diverge in the unlikely event that an existing category is disabled.

Recommendation

Consider also checking if the asset is enabled for the given category to align the implementation with Aave's `getUserAccountData()`.

Developer Response

Fixed in PR#193.

2.8.3 Insufficient precision in `categoryId` mapping for future AAVE market integrations

The `categoryId` mapping may lack precision for future AAVE market integrations on the same chain.

Technical Details

The `CollateralVaultFactory` has an internal mapping `categoryId` that can be set through `setCategoryId()` by the owner. This mapping defines which category ID new AAVE collateral vaults should use based on collateral and target asset. Category 0 uses the default LTV, while other categories enable eMode for higher LTV on specific assets.
The contract doesn't account for AAVE having multiple markets on some chains. For example, mainnet has Core, Prime, Horizon RWA, and EtherFi markets. While Twyne currently targets only Core, future support for other AAVE markets would cause the factory to use the same category ID for all collateral/target asset pairs across different markets, potentially resulting in invalid IDs or unintended categories.

Impact

Low. Future AAVE integrations on the same chain may cause issues.

Recommendation

Add a `targetVault` parameter to the mapping:

```
- mapping(address collateralAsset => mapping( address targetAsset => uint8 categoryId)) public categoryId;
+ mapping(address targetVault => mapping( address collateralAsset => mapping( address targetAsset => uint8 categoryId))) public categoryId;
```

And update the `setCategoryId()` function.

Developer Response

Fixed in PR#191.

2.9 Gas Savings Findings

2.9.1 Avoid unnecessary zero transfer in `rebalanceATokens_CV()`

Technical Details

In `rebalanceATokens_CV()`, the function transfers aTokens to the collateral vault when `shares >= actualScaledBalance`. When both values are equal, this results in a zero transfer operation.

Impact

Gas savings.

Recommendation

Replace the `else` condition with `else if (shares > actualScaledBalance)`.

Developer Response

Fixed in commit 2f0f014.

2.10 Informational Findings

2.10.1 Overridden `max*` functions break ERC-4626 specifications

Technical Details

The CustomERC4626StataTokenUpgradeable is a copy of Aave's ERC4626StataTokenUpgradeable with the `max*` functions re-defined to skip checks and allow any amount. While this saves gas, it should be noted that it breaks the ERC-4626 standard.

Impact

Informational.

Recommendation

Notify integrators about this behavior.

Developer Response

Acknowledged. This wrapper is an internal complexity.

2.10.2 Provide multiple factory functions instead of switching on VaultType

Technical Details

The implementation of `createCollateralVault()` uses a VaultType enum to determine the creation logic between Euler and Aave.
This can lead to a confusing interface, given that the creation logic differs between the two protocols. One example is the `_targetAsset` parameter, which is required for Aave but not used in Euler.

Impact

Informational.

Recommendation

Split the creation functionality between separate functions, such as `createEulerCollateralVault()` and `createAaveCollateralVault()`.

Developer Response

Acknowledged.

2.10.3 `skim()` can use the modifier `onlyBorrowerAndNotExtLiquidated`

Technical Details

Given the refactor of `_isNotExternallyLiquidated()`, the `skim()` function can avoid duplicating the validation logic, as there is no benefit in re-using cached variables.

Impact

Informational.

Recommendation

```
-   function skim() external callThroughEVC whenNotPaused nonReentrant {
-       // copied from onlyBorrowerAndNotExtLiquidated modifier to cache balanceOf
-       require(_msgSender() == borrower, ReceiverNotBorrower());
-       require(_isNotExternallyLiquidated(), ExternallyLiquidated());
+   function skim() external onlyBorrowerAndNotExtLiquidated whenNotPaused nonReentrant {
```

Developer Response

Fixed in PR#192.

2.10.4 Missing event in `setCategoryId()`

Technical Details

The `setCategoryId()` function doesn't emit an event when updating the category configuration.

Impact

Informational.

Recommendation

Emit an event in `setCategoryId()`.

Developer Response

Fixed in PR#192.

2.10.5 Update natspec documentation

Technical Details

Natspec can be updated throughout the code to improve documentation:

- Natspec for `handleExternalLiquidation()` can be changed for AAVE integration:

```
- to be called if the vault is liquidated by Euler
+ to be called if the vault is liquidated by AAVE
```

- Missing natspec for `setOracleResolvedVaultForOracleRouter()`.
- Missing `VaultType _vaultType` natspec in `createCollateralVault()`.

Impact

Informational.

Recommendation

Update the natspec.

Developer Response

Fixed in PR#192.

2.10.6 Inconsistent contract versions

Technical Details

The `CollateralVaultFactory` contract was updated to return `2` as its version, but the `VaultManager` and `EulerCollateralVault` contracts still return `1` in `version()` despite being updated since the last deployment.

Impact

Informational. Inconsistent versioning between contracts.

Recommendation

Align versions for contracts that were previously deployed and then modified.

Developer Response

Fixed in PR#192.

2.10.7 Unused `AAVE_POOL` variable

Technical Details

The variable `AAVE_POOL` is never used in the AaveV3Wrapper contract.

Impact

Informational.

Recommendation

Remove the variable.

Developer Response

Fixed in PR#192.

3 Final remarks

The review focused on the integration of Aave V3 collateral vaults and the wrapped aToken system, which extends the protocol's capabilities beyond its original Euler Finance implementation.
The codebase maintains the architectural strengths observed in prior audits. The Aave integration follows similar design patterns while adapting to Aave's specific mechanics, including E-mode categories and liquidation thresholds.
No critical or high-severity vulnerabilities were identified during this review. The wrapped aToken reward accounting issue is particularly noteworthy, although it doesn't affect the protocol's core mechanics. The original implementation assumes tokens remain within a single contract, but Twyne's design requires tokens to move between the wrapper and collateral vaults. This architectural mismatch required a rework of the reward distribution system, which ultimately led to a more straightforward implementation.
The team demonstrated exceptional responsiveness in addressing all identified issues, with the codebase showing maturity and thoughtful design decisions throughout the integration.
Fixes and code changes related to this audit were reviewed up to commit 452e2b142222b6b54ee87e0df2ad89cc755fdeea.