Electisec Twyne Review

Review Resources:

- Protocol Mechanics documentation

Auditors:

- HHK
- adriro

Table of Contents

Review Summary

Twyne

Twyne is a risk-modular credit delegation protocol built for the Ethereum Virtual Machine. It addresses a fundamental inefficiency in current DeFi lending markets: the unused borrowing power of users who deposit assets but do not borrow against them. By enabling these depositors (Credit LPs) to earn additional yield by making their unused borrowing power available to borrowers, Twyne unlocks higher capital efficiency while maintaining the security constraints of the underlying lending markets.

The contracts of the Twyne repository were reviewed over 5 days. Two auditors performed the code review between April 14th and April 18th, 2025. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review `deca583d97e11f3c0ad2140401d01a9b3f863a41`.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src
├── TwyneFactory
│   ├── BridgeHookTarget.sol
│   └── CollateralVaultFactory.sol
```

```
└── twyne
    ├── CollateralVaultBase.sol
    ├── Errors.sol
    ├── EulerCollateralVault.sol
    ├── HealthStatViewer.sol
    ├── IRMTwyneCurve.sol
    ├── VaultBase.sol
    └── VaultManager.sol
```

After the findings were presented to the Twyne team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

Electisec and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. Electisec and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Twyne and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Proper access control is in place to secure governance actions and borrower access to the collateral vault.
Mathematics	Average	Potential overflows were identified in the implementation of some calculations.
Complexity	Average	The novel mechanics and the non-standard usage of the vault increase complexity.
Libraries	Good	The implementation uses the OpenZeppelin library and integrates with Euler's EVC and EVK.
Decentralization	Good	Privileged actions are kept at a minimum to adjust protocol-wide settings. Users can unwind their position if the protocol is paused.
Code stability	Good	The codebase remained stable throughout the review.
Documentation	Good	Detailed documentation about the protocol and the mathematical foundations were presented to the auditors.
Monitoring	Low	At the start of the audit, the code was missing events, preventing accessible off-chain monitoring. This was fixed during the review, greatly improving off-chain monitoring capabilities.
Testing and verification	Good	The codebase includes a comprehensive testing suite with good coverage. The team has informed us that an invariant test suite is being developed.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings

    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

1. Medium - Vault may become unprofitable to liquidate without triggering external liquidation

A collateral vault may end up in a state where liquidating it on Twyne is not profitable but still has a high enough health factor from the target vault's perspective, which means it cannot be liquidated on the underlying protocol.

Technical Details

Twyne collateral vaults can be liquidated in two different ways:

1. The user's collateral adjusted by the Twyne LTV is less than the externally borrowed amount
2. If the vault is too close to being liquidated on the external underlying protocol.

The underlying protocol could liquidate the vault if the Twyne liquidation doesn't kick in fast enough. The protocol expects it and has a way to handle it using the `handleExternalLiquidation()` function.

However, if a user has excess credit and a high Twyne LTV, when the collateral value drops because of market volatility, the Twyne liquidation may not kick in fast enough. Still, the vault's health factor may stay high enough that the external liquidation may not kick in either.

Example:

LTVtwyne=95% LTVLiquidationAave=80% LTVBorrowAave=78% Buffer=98%
Deposit=100$ reserve=30$ total=130$ borrow=95$

Collateral vault: 100 * .95 >= 95 -> can't be liquidated
Target vault: 130 * (80% * 98%) > 95 -> can't be externally liquidated

The reserve should be around ~22$, but we borrowed a little over (excess credit).
No one calls `rebalance()`, and the price drops by 5%.

Deposit=95$ reserve=28.5$ total=123.5$ borrow=95$

Collateral vault: 95 * .95 < 95 -> can be liquidated but liquidator **won't make a profit**
123.5 * (80% * 98%) > 95 -> can't be externally liquidated

We end up in a case where the vault is between both liquidations. The lenders can't call `rebalance()` at this point, as releasing assets from the collateral vault will result in a negative health factor position from the external protocol's perspective, meaning the call will revert. So, we have to wait for the collateral value to increase. Hence, it becomes profitable to liquidate on Twyne or to decrease enough for the external liquidation to trigger, but at the cost of the intermediate vault lenders.

Impact

Medium. Edge cases may happen with liquidations where a vault is not worth liquidating internally, cannot be liquidated externally, and rebalancing is impossible.

Recommendation

Consider adding `rebalance()` inside the vault check so that users don't take more credit than needed, as it will be reset at the end of every vault interaction.

Developer Response

Fixed in PR #14 & PR #16 & PR #17.

Low Findings

1. Low - The unit of account for the intermediate and target vaults could differ

The implementation of `_canLiquidate()` compares two values that could potentially be quoted in a different base asset.

Technical Details

In the following expression, `externalBorrowDebtValue` is quoted in the target vault's unit of account, and `userCollateralValue` is quoted in the intermediate vault's unit of account.

```
        return (externalBorrowDebtValue * MAXFACTOR > twyneLiqLTV *
userCollateralValue);
```

Impact

Low. The check will be incorrect if the vaults are set up with a different unit of account.

Recommendation

The VaultManager.sol contract could check that both vaults have the same unit of account by adding the corresponding validations in `setIntermediateVault()` and `setAllowedTargetVault()`.

Developer Response

Fixed in PR #4.

2. Low - Creation salt is always derived from the EVC account in CollateralVaultFactory.sol

Since the `createCollateralVault()` function requires to be proxied through the EVC, the actual caller will always be the EVC account.

Technical Details

The implementation of `createCollateralVault()` uses `msg.sender` to derive the salt used while creating the BeaconProxy with CREATE2.

```
76:         vault = address(new BeaconProxy{salt:
keccak256(abi.encodePacked(msg.sender, nonce[msg.sender]++))}
(collateralVaultBeacon[_targetVault], ""));
```

Impact

Low. Vault addresses are derived from a unique account and could lead to race conditions and incorrect predictions.

Recommendation

Use `_msgSender()` instead of `msg.sender` to correctly use the address of the actual caller of the function.

Developer Response

Fixed in PR #6.

3. Low - Interest accumulation can lead to underflow when computing the user's collateral

Multiple functions inside the codebase use `maxRelease()` and subtract it from `totalAssetsDepositedOrReserved`. However, this subtraction could lead to an underflow if interest accumulates for a long time.

Technical Details

The functions `_canRebalance()`, `_canLiquidate()` and `_hasNonNegativeExcessCredit()` compute the user's collateral with `totalAssetsDepositedOrReserved - maxRelease()`.

However, `maxRelease()` could return a value greater than `totalAssetsDepositedOrReserved` if interest has accumulated for a long time and no rebalancing or liquidation occurred.

If lenders do not monitor the vault and if the liquidation is not profitable enough for no liquidator to execute it, the vault could end up in a state where it's impossible for lenders to get back their assets.

Impact

Low. While unlikely, there is a scenario in which the `maxRelease()` can become greater than `totalAssetsDepositedOrReserved` and DoS the vault, locking in the lender's assets.

Recommendation

Consider checking if `maxRelease() > totalAssetsDepositedOrReserved` and assume that the user's collateral is zero in that case.

Developer Response

Fixed in PR #7.

4. Low - `splitCollateralAfterExtLiq()` could underflow

Technical Details

The function `splitCollateralAfterExtLiq()` computes the `releaseAmount` by subtracting `liquidatorReward` from the `_collateralBalance`.

However, in some rare cases where the target vault LTV is high and the bonus received by the liquidator added to the `liquidatorReward` is also high, there could be a case where `liquidatorReward > _collateralBalance` which would lead to an underflow.

Impact

Low. While unlikely, there is a scenario where the `liquidatorReward` can become greater than `_collateralBalance` and cause a revert condition.

Recommendation

Consider capping the `liquidatorReward` at the `_collateralBalance` so that it is not greater than the available collateral.

Developer Response

Fixed in PR #9.

5. Low - Teleport can be used to break external liquidation invariant

An incorrect assumption in the `teleport()` function could disrupt the equality between asset balance and internal assets tracking, allowing an external liquidation to be triggered.

Technical Details

The teleport feature allows users to migrate their debt position to Twyne from the external protocol by bundling several actions. In particular, the borrower can specify an amount (`toReserve`) to pull funds from the credit vault. These assets are borrowed from the intermediate vault and tracked in `totalAssetsDepositedOrReserved`.

```
        totalAssetsDepositedOrReserved += toDeposit + toReserve;
        intermediateVault.borrow(toReserve, address(this));
```

The issue here is incorrectly assuming that `toReserve` is the amount of assets being transferred by `borrow()`. Euler vaults support an argument of `type(uint256).max`, which [transfers all of the available cash](). This means that `totalAssetsDepositedOrReserved` can be updated by `type(uint256).max. In contrast, the amount of assets sent to the collateral vault will differ.

Impact

Low. While no direct attack path was identified, allowing a forced external liquidation scenario might lead to unexpected states that could enable a potential exploit, such as intentionally causing bad debt to the credit vault.

Recommendation

Update `totalAssetsDepositedOrReserved` by the `borrow()` function return value.

```
-   totalAssetsDepositedOrReserved += toDeposit + toReserve;
-   intermediateVault.borrow(toReserve, address(this));
+   totalAssetsDepositedOrReserved += toDeposit +
intermediateVault.borrow(toReserve, address(this));
```

Developer Response

Fixed in [PR #10]().

Gas Saving Findings

1. Gas - Gas optimization when deploying a new vault

Technical Details

When deploying a new vault, some elements can be removed or optimized to save some gas for the users:

- Inside `__CollateralVaultBase_init()` the `forceApprove()` can take `__asset` as a parameter instead of the storage variable `_asset`.
- Inside `__CollateralVaultBase_init()` the `require()` at the end of the function is redundant. The `VaultManager` contract already ensures that the `intermediateVault` has the same asset as the collateral vault.
- Inside `__CollateralVaultBase_init()` the `intermediateVault` could be cached when queried from `twyneVaultManager.getIntermediateVault()` as it’s used multiple times.
- Inside `_depositUnderlying()` the `forceApprove()` could be moved to the vault deployment so it doesn’t approve on every deposit.

Impact

Gas savings.

Recommendation

Apply the changes suggested.

Developer Response

Fixed in PR #8 & PR #18.

2. Gas - Unnecessary check in `checkLiqLTVByCollateralVault()`

Technical Details

The implementation of `checkLiqLTVByCollateralVault()` checks that the caller is a registered collateral vault.

Since this is a view function that can’t affect the state, there is no point in doing access control. Also note that the implementation needs to re-query the caller to fetch the target vault and asset, which can be directly passed as arguments by the caller (the collateral vault).

Impact

Gas savings.

Recommendation

Remove the check.

Additionally, the function signature is a bit confusing since it returns a bool indicating the result of the check, but internally, it reverts and never returns false. The semantics can be changed so that `checkLiqLTV()` returns true or false without reverting, and `checkLiqLTVByCollateralVault()` reverts without returning a result.

Developer Response

Fixed in PR #12.

3. Gas - Simplify `internalHF()` implementation

Technical Details

Inside the helper function `internalHF()` the path to get the `intermediateVault` could be simplified with `collateralVault.intermediateVault()` to save gas.

Impact

Gas savings.

Recommendation

Update the function with the change suggested.

Developer Response

Fixed in PR #11.

Informational Findings

1. Informational - Incorrect name and symbol initialization

The collateral vault's name and symbol are initialized in the constructor of the implementation.

Technical Details

CollateralVaultBase.sol#L55

```
55:     constructor(address _evc, address _targetVault) VaultBase(_evc) {
56:         targetVault = _targetVault;
57:         name = "Collateral Vault";
58:         symbol = "CV";
59:     }
```

Impact

Informational.

Recommendation

Move the initialization of `name` and `symbol` to the `__CollateralVaultBase_init()` function.

Developer Response

Fixed in PR #3.

2. Informational - Missing ReentrancyGuardUpgradeable initialization

The ReentrancyGuardUpgradeable mixin is never initialized in the VaultBase.sol contract.

Technical Details

VaultBase.sol#L16-L17

Impact

Informational.

Recommendation

Initialize the library as part of the initialization process in `__VaultBase_init()`.

Developer Response

Fixed in PR #5.

3. Informational - Make `maxTwyneLiqLTV` and `externalLiqBuffer` mappings

Technical Details

The variables `maxTwyneLiqLTV` and `externalLiqBuffer` are currently protocol-wide parameters, while Twyne is supposed to be a protocol with more risk than a classic lending market. Some assets are much more volatile than others. A drop in price too fast for Twyne liquidators to act could lead to external liquidation, resulting in a loss for credit lenders.

For these reasons, it might be safer and easier to have `maxTwyneLiqLTV` and `externalLiqBuffer` parameters per asset.

Impact

Informational.

Recommendation

Change these two variables to be a mapping between the asset and the value.

Developer Response

Fixed in PR #13.

Final Remarks

The Twyne protocol introduces innovative possibilities to the DeFi landscape. Its codebase is generally straightforward, except for the enforced excess credit invariant. The architecture is intentionally optimized around this invariant. It leverages the Euler Finance EVC and EVK frameworks to minimize code duplication and benefit from the security of a well-tested foundation. No high or critical severity vulnerabilities were identified during our review, reinforcing the overall soundness of the design. The team has demonstrated strong responsiveness and efficiency in addressing reported issues and implementing suggested improvements. The team provided extra fixes that were not directly linked to findings, and these commits were also reviewed, up to commit `c246f25f38225f9c27c98fad305d3465ac6b3ef0`.