yAudit

November 24, 2025

Prepared for
Twyne

Audited by
HHK
adriro

Twyne Incremental Review

Smart Contract Security Assessment

Contents

1 Review Summary **2**
- 1.1 Protocol Overview . . . 2
- 1.2 Audit Scope . . . 2
- 1.3 Risk Assessment Framework . . . 2
  - 1.3.1 Severity Classification . . . 2
- 1.4 Key Findings . . . 3
- 1.5 Overall Assessment . . . 3

2 Audit Overview **3**
- 2.1 Project Information . . . 4
- 2.2 Audit Team . . . 4
- 2.3 Audit Resources . . . 4
- 2.4 Critical Findings . . . 4
- 2.5 High Findings . . . 4
  - 2.5.1 Teleport should verify subaccount is tied to current borrower . . . 4
- 2.6 Medium Findings . . . 5
- 2.7 Low Findings . . . 5
  - 2.7.1 Missing SafeERC20 operations . . . 5
- 2.8 Gas Savings Findings . . . 6
  - 2.8.1 Gas improvements in EulerWrapper . . . 6
- 2.9 Informational Findings . . . 6
  - 2.9.1 Unchecked approval parameter inside `EulerWrapper` . . . 6
  - 2.9.2 Ineffective locking mechanism in LeverageOperator . . . 7
  - 2.9.3 Unused variable in LeverageOperator . . . 8
  - 2.9.4 Empty contracts in deployment script . . . 8
- 2.10 Final Remarks . . . 9

1 Review Summary

1.1 Protocol Overview

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.

1.2 Audit Scope

This audit covers the incremental review for Twyne V1 in PR 35 to the smart contracts in scope across 1.5 days of review.

```
src/
├── Periphery
│   └── EulerWrapper.sol
├── TwyneFactory
│   └── CollateralVaultFactory.sol
├── operators
│   └── LeverageOperator.sol
└── twyne
    ├── CollateralVaultBase.sol
    ├── EulerCollateralVault.sol
    └── VaultManager.sol
script
└── TwyneDeployEulerIntegration.s.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.	Reduced transaction costs
Informational	Code quality and best practice recommendations	Improved maintainability and readability

Table 1: severity classification

1.4 Key Findings

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	1
Medium	0
Low	1
Informational	4

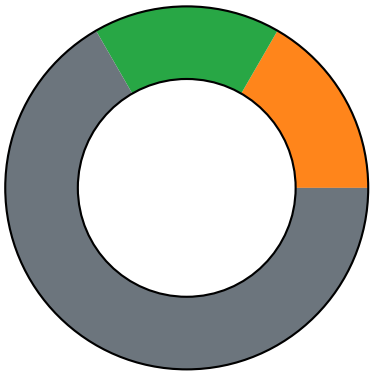

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

2 Audit Overview

2.1 Project Information

Protocol Name: Twyne
Repository: https://github.com/0xTwyne/twyne-contracts-v1
Commit Hash: 802103a5b6621b64dea9adec43529728d7a0e374
Commit URL: https://github.com/0xTwyne/twyne-contracts-v1/commit/802103a5b6621b64dea9adec43529728d7a0e374

2.2 Audit Team

HHK, adriro

2.3 Audit Resources

Code repositories and documentation

2.4 Critical Findings

None.

2.5 High Findings

2.5.1 Teleport should verify subaccount is tied to current borrower

A missing validation could be used to pull funds from dangling authorization.

Technical Details

The updated functionality of `teleport()` can be used to migrate a position from a subaccount of the borrower.

```
237:     function teleport(uint toDeposit, uint toBorrow, address subAccount) external
onlyBorrowerAndNotExtLiquidated whenNotPaused nonReentrant {
238:         createVaultSnapshot();
239:
240:         totalAssetsDepositedOrReserved += toDeposit;
241:         _handleExcessCredit(_invariantCollateralAmount());
242:
243:         if (toBorrow == type(uint).max) {
244:             toBorrow = IEVault(targetVault).debtOf(subAccount);
245:         }
246:
247:         IEVC.BatchItem[] memory items = new IEVC.BatchItem[](3);
248:         items[0] = IEVC.BatchItem({
249:             targetContract: asset(),
250:             onBehalfOfAccount: address(this),
251:             value: 0,
252:             data: abi.encodeCall(IERC20.transferFrom, (subAccount, address(this),
toDeposit)) // needs allowance
253:         });
```

The problem is the absence of a validation that ties the subaccount to the current borrower, leading to a `transferFrom()` action from an arbitrary account.
Typically, there should be no active allowance from third-party accounts, as collateral vaults are allocated per borrower. However, since vaults can be liquidated, the borrower may shift to the new liquidator, enabling the new borrower to access the old borrower's funds.

Impact

High. New vault owners can siphon funds from a previous borrower.
Requirements:

- Collateral vault gets liquidated
- Excess approval exists
- The previous vault owner contains a non-zero amount of the collateral token
- Funds can only be pulled from the previous vault owner

Recommendation

Validate `subaccount` is an actual subaccount of the current `borrower`.

Developer Response

Fixed in commit 5945b31. To clarify, the proper implementation already existed in the MockCollateralVault contract.

2.6 Medium Findings

None.

2.7 Low Findings

2.7.1 Missing SafeERC20 operations

The LeverageOperator contract executes ERC20 operations over arbitrary tokens without the SafeERC20 wrapper, potentially causing incompatibility issues.

Technical Details

- LeverageOperator.sol#L156
- LeverageOperator.sol#L195

Impact

Low.

Recommendation

Use `safeTransfer()` and `forceApprove()`.

Developer Response

Fixed in PR#36.

2.8 Gas Savings Findings

2.8.1 Gas improvements in EulerWrapper

Technical Details

The EulerWrapper contract enables a zapper functionality to deposit in Twyne.
Both functions have the `callThroughEVC` modifier, which will re-route the call through the EVC. This modifier shouldn't be needed as the implementation doesn't strictly require it.
Additionally, note that each call to EVK vaults would need to be re-wrapped in the EVC that corresponds to each vault (which may be Euler's or Twyne's).
Additionally, the `pull token → approve → deposit` cycle can leverage the *skim* functionality and instead do:
1) transfer tokens from the caller to collateral 2) call `skim()` on the collateral with the intermediate vault as the recipient 3) call `skim()` on the intermediate vault with the user as the recipient

Impact

Gas savings.

Recommendation

Consider applying the suggested modifications.

Developer Response

Fixed in PR#36.

2.9 Informational Findings

2.9.1 Unchecked approval parameter inside `EulerWrapper`

Technical Details

The function `depositUnderlyingToIntermediateVault()` takes an `intermediateVault` parameter and approves the Euler asset on it.

However, this `intermediateVault` is not enforced to be a legit vault deployed by the factory. This allows any user to approve a malicious contract.
While the contract is not supposed to hold funds, it is advised to check the `intermediateVault` against the Vault manager.

Impact

Informational.

Recommendation

Ensure the `intermediateVault` is a legitimate vault against the Vault Manager and/or implement the gas-finding recommendations, which will remove the `approval()` call.

Developer Response

Fixed in PR#36.

2.9.2 Ineffective locking mechanism in LeverageOperator

The mechanism provided by `flashloanLock` does not enforce any lock.

Technical Details

The `flashloanLock` is toggled before executing the flashloan in `executeLeverage()`. The implementation doesn't provide any actual locking benefits because:

- This doesn't work as a reentrancy guard: the function can be re-entered while the flag is on.
- `onMorphoFlashLoan()` cannot be executed by anything other than the LeverageOperator itself, cause the caller must be Morpho, and Morpho only calls the flashloan initiator.

Impact

Informational. Gas savings.

Recommendation

Change the semantics of this lock to be a normal reentrancy guard in the `executeLeverage()` function or remove the `flashloanLock` variable.

Developer Response

`flashloanLock` variable removed in PR#36. Added nonreentrant modifier for extra safety, even if it is not necessarily required.

2.9.3 Unused variable in LeverageOperator

The `initialCollateralBalance` variable is written but never read.

Technical Details

LeverageOperator.sol#L40.

Impact

Informational.

Recommendation

Remove the variable.

Developer Response

Fixed in PR#36.

2.9.4 Empty contracts in deployment script

The contracts referenced in the `productionSetup()` function are empty accounts in Ethereum.

Technical Details

TwyneDeployEulerIntegration.s.sol#L280-L284.

```
280:         } else if (block.chainid == 1) {
281:             oracleRouterFactory = 0x72735e5dd42EDc979c600766532eA704842CfB7b;
282:             evc = EthereumVaultConnector(payable(0
xC36aED7b7816aA21B660a33a637a8f9B9B70ad6c));
283:             factory = GenericFactory(0xd5e966dB359f1cB2A01280fCCBEB839Ac572CE35);
284:             protocolConfig = ProtocolConfig(0
x3b68711EF6c1988c96CBD32d929b76cB09b579Ea);
```

Impact

Informational.

Recommendation

Ensure to deploy these contracts properly before executing the script.

Developer Response

Acknowledged. These addresses are just placeholders. The deployment process involves executing an evk-periphery script, which provides these addresses.

2.10 Final Remarks

The Twyne protocol maintains a solid foundation built on Euler Finance's EVC and EVK frameworks. This review identified one high-severity issue in the `teleport()` function, which was promptly fixed by the team during the review. Afterward, a new scope was submitted and reviewed.
The codebase remains straightforward with clear architecture. The team demonstrated strong responsiveness in addressing all findings and implementing improvements, reinforcing confidence in the protocol's development practices.
The team provided extra fixes inside PR#36 that were not directly linked to findings, and these commits were also reviewed, up to commit 1f7e12fd5aeb196a57691adec5a4396214c419cb which corresponds to the merge of PR#35 and PR#36 into the main branch.