ENIGMA DARK

Securing the Shadows

Invariant Testing Engagement

Twyne v1

April, 2025

Contents

Summary

Enigma Dark
Enigma Dark is a web3 security firm leveraging the best talent in the space to secure all kinds of blockchain protocols and decentralized apps. Our team comprises experts who have honed their skills at some of the best auditing companies in the industry. With a proven track record as highly skilled white-hats, they bring a wealth of experience and a deep understanding of the technology and the ecosystem.

Learn more about us at enigmadark.com

Twyne v1
Twyne is a credit delegation protocol that turns unused borrowing power into yield. It maximizes lending efficiency by delegating unused borrowing power to users that want to increase leverage or protect against liquidation.

Engagement Overview

Over the course of 3 weeks, beginning March 31 2025, the Enigma Dark team conducted a continous invariant testing engagement of the Twyne v1 project. The engagement was performed by one Lead Security Researcher: vnmrtz.

The following repositories were reviewed at the specified commits:

Repository	Commit
0xTwyne/twyne-contracts-v1	`deca583d97e11f3c0ad2140401d01a9b3f863a41`

Risk Classification

Severity	Description
Critical	Vulnerabilities that lead to a loss of a significant portion of funds of the system.
High	Exploitable, causing loss or manipulation of assets or data.
Medium	Risk of future exploits that may or may not impact the smart contract execution.
Low	Minor code errors that may or may not impact the smart contract execution.
Informational	Non-critical observations or suggestions for improving code quality, readability, or best practices.

Vulnerability Summary

Severity	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	1	1	0
Low	2	0	2
Informational	1	1	0

Invariants and Postconditions Summary

This table provides an overview of all invariants and postconditions defined in the suite specification files.

Property Types

On this invariant testing framework there exists two types of Properties:

INVARIANTS (INV)

- These are properties that should always hold true in the system
- They are implemented under `/invariants` folder

POSTCONDITIONS

- These are properties that should hold true after an action is executed
- They are implemented under `/hooks` and `/handlers` folders
- There are two types of POSTCONDITIONS:

GLOBAL POSTCONDITIONS (GPOST)

- These are properties that should always hold true after any action is executed
- They are checked in the `_checkPostConditions` function within the HookAggregator contract

HANDLER-SPECIFIC POSTCONDITIONS (HSPOST)

- These are properties that should hold true after a specific action is executed in a specific context
- They are implemented within each handler function, under the HANDLER-SPECIFIC POSTCONDITIONS section

Table Structure

- **Property ID**: The unique identifier for each property
- **Description**: The detailed explanation of what the property should ensure
- **Status**: Current testing status (PASS/FAIL/PENDING)
- **Issues**: Related issues or notes when status is not PASS

Property ID	Description	Status	Issues
INV_CV_BASE_A	The user-set LTV must be within protocol bounds	FAIL	L-02
INV_CV_BASE_B	A collateral vault must have a non-zero borrower unless the vault has been externally liquidated and the handleExternalLiquidation function has been called	PASS	-
INV_CV_ASSETS_A	If twyne collateral vault has not been externally liquidated, its collateral asset balance must be greater or equal than totalAssetsDepositedOrReserved	PASS	-
INV_CV_ASSETS_B	If twyne collateral vault balance is less than totalAssetsDepositedOrReserved it has been externally liquidated but handleExternalLiquidation has not been called	PASS	-
INV_CV_LTV_A	Twyne LTV >= Ext LTV	PASS	-
INV_CV_DEBT_A	If maxRepay() returns non-zero, there must be external debt	PASS	-
INV_CV_LQ_A	If the collateralVault position is liquidatable it must be liquidatable on twyne	PASS	-
INV_CV_NR_A	balanceOf(borrower) must never revert	PASS	-
INV_CV_NR_B	maxRepay must never revert	PASS	-
INV_CV_NR_C	maxRelease must never revert	PASS	-
INV_CV_NR_D	canLiquidate must never revert	FAIL	M-01
INV_CV_NR_E	isExternallyLiquidated must never revert	PASS	-

Property ID	Description	Status	Issues
HSPOST_CV_BASE_A	After rebalancing, the vault should not become liquidatable if it wasn't before	PASS	-
HSPOST_CV_BASE_B	Rebalancing should never decrease user's actual collateral amount	PASS	-
HSPOST_CV_BASE_C	After rebalancing, the amount of excess credit should be 0	PASS	-
HSPOST_CV_ASSETS_A	After depositing, totalAssetsDepositedOrReserved must increase at least by deposit amount	PASS	-
HSPOST_CV_ASSETS_B	After withdrawing, totalAssetsDepositedOrReserved must decrease at least by withdrawn amount	PASS	-
HSPOST_CV_ASSETS_C	After depositing, vault balance must increase at least by deposit amount	PASS	-
HSPOST_CV_ASSETS_D	After withdrawing, vault balance must decrease at least by withdrawn amount	PASS	-
HSPOST_CV_DEBT_A	After deposit, maxRepay (credit) must increase	PASS	-
HSPOST_CV_DEBT_B	After withdraw, maxRepay (credit) must decrease	PASS	-
HSPOST_CV_DEBT_C	After borrowing, the correct amount of debt on twyne collateral vault must be created	PASS	-

Property ID	Description	Status	Issues
HSPOST_CV_DEBT_D	After repaying, the correct amount of debt on twyne collateral vault must be repaid	PASS	-
HSPOST_CV_DEBT_G	After borrowing, the correct amount of debt assets must be transferred to receiver	PASS	-
HSPOST_CV_DEBT_H	After repaying, the correct amount of debt assets must be transferred from the borrower	PASS	-
HSPOST_CV_DEBT_I	A borrower should always be able to withdraw all if there is no outstanding debt and the vault is not externally liquidated	PASS	-
HSPOST_CV_DEBT_J	A user with enough funds can always repay debt in full	PASS	-
GPOST_CV_HEALTH_A	Only a deposit, depositUnderlying, repay and handleExternalLiquidation, can leave it in an unhealthy state, and only if the borrower is already in an unhealthy state	PASS	-
GPOST_CV_HEALTH_B	Only liquidations can deteriorate health score of an already unhealthy borrower	PASS	-
GPOST_CV_HEALTH_C	After any action against the twyne collateral vault, excess credit should never be negative	PASS	-
GPOST_CV_HEALTH_D	Borrower position against twyne collateral vault should not not be liquidatable after any action	PASS	-
GPOST_CV_HEALTH_E	Unhealthy borrowers can not borrow	PASS	-

Property ID	Description	Status	Issues
GPOST_CV_SIPHON_A	totalAssetsDepositedOrReserved - maxRelease() should never increase without explicit user deposits	PASS	-
HSPOST_CV_LQ_A	Liquidation can only succeed if violator is unhealthy	PASS	-
HSPOST_CV_LQ_B	External liquidation should only be possible if the underlying position has been liquidated and vault balance < totalAssetsDepositedOrReserved	PASS	-
HSPOST_CV_LQ_C	If the underlying position has been liquidated, external liquidation should always succeed and vault balance < totalAssetsDepositedOrReserved	PASS	-
HSPOST_CV_LQ_E	After handleExternalLiquidation, totalAssetsDepositedOrReserved should be 0	PASS	-
HSPOST_CV_LQ_F	After handleExternalLiquidation, maxRepay() should return 0	PASS	-
HSPOST_CV_NR_B	If canRebalance returns a non-zero value, rebalance must not revert the vault is not liquidatable	PASS	-
HSPOST_CV_NR_D	repay(maxRepay) must never revert	FAIL	L-01

Summary

- **Total Properties**: 40
- **Invariants (INV)**: 12
- **Global Postconditions (GPOST)**: 6
- **Handler-Specific Postconditions (HSPOST)**: 22
- **Status Distribution**:
  - PENDING: 0
  - PASS: 37
  - FAIL: 3
- **Resolution Distribution** (for FAIL properties):
  - FIXED: 1
  - ACKNOWLEDGED: 2

Categories

Base Functionality

- User-set LTV bounds validation
- Borrower existence validation
- Rebalancing effects on liquidation status and collateral amounts

Asset Management

- Vault balance vs totalAssetsDepositedOrReserved consistency
- Proper tracking of deposits and withdrawals
- Collateral value preservation

Loan-to-Value (LTV) Ratios

- Relationship between Twyne LTV and external LTV

Debt Management

- Debt existence validation
- Credit tracking and excess credit bounds
- Borrowing and repayment correctness
- Asset transfer validation

Health and Liquidation

- Position health maintenance
- Liquidation conditions and effects
- External liquidation handling

- Borrower state after liquidation

Non-Revert Guarantees

- View functions must never revert
- Critical operations must succeed under valid conditions

Siphoning Protection

- Prevention of unauthorized asset extraction

Status Definitions

- **PENDING**: Property has not been tested yet or testing is in progress
- **PASS**: Property passes all tests and behaves as expected
- **FAIL**: Property fails tests and requires attention

Resolution Options (for FAIL status properties)

- **FIXED**: The underlying issue has been resolved by the protocol team, property should pass on re-testing
- **ACKNOWLEDGED**: The issue has been reviewed and acknowledged by the protocol team as acceptable risk/design decision

*Note: Detailed explanations for all failing properties and their related issues can be found in the **Findings** section below.*

Testing Workflow

1. Properties start as **PENDING** before testing
2. After initial testing, properties become **PASS** or **FAIL**
3. For **FAIL** status properties, the protocol team responds with either:
    - **FIXED**: Issue gets resolved (property should pass on re-testing)
    - **ACKNOWLEDGED**: Issue is accepted as acceptable risk/design decision

Findings

Index	Issue Title	Status
M-01	Denial of service in liquidations if collateral vault remains open long enough	Fixed
L-01	Max repayment may fail after decrease in collateral asset price	Acknowledged
L-02	Increasing external liquidation buffer can violate collateral vault’s minimum LTV	Acknowledged
I-01	Minor improvements to code and comments	Fixed

Detailed Findings

High Risk

No issues found.

Medium Risk

M-01 - Denial of service in liquidations if collateral vault remains open long enough

Severity: Medium Risk

Context:

- EulerCollateralVault.sol#L116
- EulerCollateralVault.sol#L143

Technical Details:
In lending protocols, it is critical to maintain the availability property that ensures liquidations are always available for underwater positions. In the case of Twyne invariant suite, this is implemented with the invariant `INV_CV_NR_D: canLiquidate must never revert`.

However, in Twyne's implementation, if a collateral vault debt position remains open long enough, the debt owed to the intermediate vault can eventually grow larger than `totalAssetsDepositedOrReserved`. When this happens, the calculation `totalAssetsDepositedOrReserved - maxRelease()` underflows and causes the transaction to revert.

Impact:
If the collateral vault is left unchecked over time, liquidations can become unavailable, potentially leading to the accumulation of bad debt in the system.

Recommendation:
Consider updating `maxRelease` to return the minimum of the intermediate vault's debt and `totalAssetsDepositedOrReserved` to prevent the underflow condition.

Developer Response:
Fixed at commit `4888034`.

`maxRelease` now correctly returns the minimum value between `intermediateVault.debtOf(address(this))` and `totalAssetsDepositedOrReserved`, ensuring that the `INV_CV_NR_D` invariant consistently holds.

Low Risk

L-01 - Max repayment may fail after decrease in collateral asset price

Severity: Low Risk

Context:

- EulerCollateralVault.sol#L65-L69
- CollateralVaultBase.sol#L176-L187

Technical Details:
In lending protocols, it is critical to maintain the availability property that ensures users and integrators can always repay the maximum allowable amount without failure. Specifically, the invariant `HSPOST_CV_NR_D: repay(maxRepay) must never revert` must consistently hold.

However, in the case of Twyne, this invariant breaks when the value of the collateral drops below a certain threshold.

PoC:

```
    // HSPOST_CV_NR_D

    function assert_HSPOST_CV_NR_D() external
setupActor(collateralVaultUser) {
        bool success;
        bytes memory data;

        uint256 maxRepay = userCollateralVault.maxRepay();
        require(maxRepay != 0, "maxRepay should not be 0");

        uint256 collateralBalanceOfBorrower =
assetTST.balanceOf(collateralVaultUser);

        // Ensure borrower has enough assets for repayment
        if (collateralBalanceOfBorrower < maxRepay) {
            assetTST.mint(collateralVaultUser, maxRepay -
collateralBalanceOfBorrower);
        }

        target = address(userCollateralVault);

        _before();
        (success, data) = actor.proxy(target,
abi.encodeCall(CollateralVaultBase.repay, maxRepay));

        assertTrue(success, HSPOST_CV_NR_D);

        _after();
    }
```

```
    // Replay test

    function test_replay_assert_HSPOST_CV_NR_D() public {
        Tester.mint(2739, 6, 1);
        Tester.setPrice(100000, 0);
        Tester.mint(1, 0, 0);
        Tester.deposit(1174, 0, 2);
        Tester.depositCV(12);

Tester.borrowCV(115792089237316195423570985008687907853269984665640564039457
584007913129639935, 0);
        Tester.setPrice(1, 1);
        Tester.assert_HSPOST_CV_NR_D();
    }
```

Impact:
Users and integrators are unable to repay the maxRepay amount if the value of collateral is too low.

Recommendation:
Add additional logic to handle the edge case of large price drops in collateral assets to avoid this revert case.

Developer Response:
Acknowledged. Our plan is to check all the operations on twyne when collateral asset price drops to a very low value. We'll compare the execution against our expectation. Our immediate reponse to repay() failing is: if collateral is worthless, no rational borrower will repay the debt. We can change the invariant for now to avoid this case.

Twyne will only support blue chip assets as collateral for the short term, which makes this a highly unlikely scenario.

L-02 - Increasing external liquidation buffer can violate collateral vault's minimum LTV

Severity: Low Risk

Context:

- VaultManager.sol#L98-L101

Technical Details:
The `VaultManager` contract allows governance to update the `externalLiqBuffer` for a collateral asset via the `setExternalLiqBuffer` function. However, in certain scenarios, increasing this buffer can cause the collateral vault's Loan-to-Value (LTV) ratio to fall below the allowed minimum, violating system constraints and making the vault liquidatable.

PoC:

```
    // INV_CV_BASE_A

    function assert_INV_CV_BASE_A() internal {
        uint256 liqLTV = userCollateralVault.twyneLiqLTV();
        uint256 minLTV = eTST.LTVLiquidation(address(eTST2));
        uint256 externalLiqBuffer =
vaultManager.externalLiqBuffers(address(eTST2));
        uint256 maxTwyneLiqLTV = vaultManager.maxTwyneLTVs(address(eTST2));

        assertGe(liqLTV * 1e4, minLTV * externalLiqBuffer, INV_CV_BASE_A);
        assertLe(liqLTV, maxTwyneLiqLTV, INV_CV_BASE_A);
    }
```

```
    // Replay test

    function test_replay_setExternalLiqBuffer() public {
        _setUpActor(USER1);
        Tester.setExternalLiqBuffer(0);
        Tester.setTwyneLiqLTV(8585);
        Tester.setExternalLiqBuffer(40);
        assert_INV_CV_BASE_A();
    }
```

Impact:
Collateral vault's LTV can go below its allowed minimum.

Recommendation:
Introduce an additional validation in the `setExternalLiqBuffer` function to ensure that increasing the `externalLiqBuffer` does not reduce the vault's LTV below the allowed minimum. The function should revert if this condition is not met.

Developer Response:
Acknowledged. It may be better to leave this test as is and and expect a revert. During the whitelist period, we'll discuss with the team if some mechanism to handle this change in external liquidation buffer is needed.

Most lending protocols have a similar situation where governance can change the allowed LTV values to a point which can cause some users to be liquidated

Informational

I-01 - Minor improvements to code and comments

Severity: Informational

Context: See below.

Technical Details:

1. EulerCollateralVault.sol#L86 - The NatSpec comment incorrectly uses the word “help” instead of “held” when referring to the collateral assets.
2. CollateralVaultBase.sol#L385-L388 - Consider caching the result of `_invariantCollateralAmount` in a local variable to avoid redundant external calls.

Developer Response:
Fixed at commit `ff27a9e` .

Disclaimer

This report does not endorse or critique any specific project or team. It does not assess the economic value or viability of any product or asset developed by parties engaging Enigma Dark for security assessments. We do not provide warranties regarding the bug-free nature of analyzed technology or make judgments on its business model, proprietors, or legal compliance.

This report is not intended for investment decisions or project participation guidance. Enigma Dark aims to improve code quality and mitigate risks associated with blockchain technology and cryptographic tokens through rigorous assessments.

Blockchain technology and cryptographic assets inherently involve significant risks. Each entity is responsible for conducting their own due diligence and maintaining security measures. Our assessments aim to reduce vulnerabilities but do not guarantee the security or functionality of the technologies analyzed.

This security engagement does not guarantee against a hack. It is a review of the codebase during a specific period of time. Enigma Dark makes no warranties regarding the security of the code and does not warrant that the code is free from defects. By deploying or using the code, the project and users of the contracts agree to use the code at their own risk. Any modifications to the code will require a new security review.