ENIGMA DARK

Securing the Shadows



Managed Security Review

Alto Lending

July 25, 2025

Contents

1. Summary
2. Engagement Overview
3. Risk Classification
4. Vulnerability Summary
5. Findings
6. Disclaimer

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

Alto Lending
Alto Lending is a platform with isolated markets that enable borrowing and lending using the stablecoin USDO. It includes traditional supply-based markets and mint-based markets, with adjustable interest rates and a partial liquidation system to manage risk.

Engagement Overview

Over the course of 1 week, beginning June 25 2025, Enigma Dark coordinated a Managed Security Review of the Alto Lending project. The review was performed by one Security Researcher: kiki.

The following repositories were reviewed at the specified commits:

Repository	Commit
Tapioca-DAO/tapioca-protocol	`cd400bfc868a5880a0c0336086e13288f5e61083`

The scope of the review covered the following files:

```
contracts
├── lending
|   ├── AltoBorrowMarket
|   ├── AltoMintMarket
|   ├── irm/
|   ├── leverage/
|   └── libraries/
└── utils
    ├── AltoAdapter
    ├── Auth
    ├── AuthUpgradeable
    ├── FixedPointMath
    └── Pausable
```

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	0	1
Low	7	6	1
Informational	5	5	0

Findings

Index	Issue Title	Status
M-01	Liquidation Fee Prioritization Leads to Bad Debt	Acknowledged
L-01	Grace Period Bypassed From Persisting Data	Fixed
L-02	Protocol Fee Lost to Zero Address Recipient	Acknowledged
L-03	Fee Recipient Lacks Zero Address Validation	Fixed
L-04	Borrow Fees Round Down Instead of Up Against Users	Fixed
L-05	Calling `setIrm` Will Accrue Interest While Market Paused	Fixed
L-06	Debt Ceiling Changes Accrue Interest While Paused	Fixed
L-07	Taylor Series Approximation Loses Precision at High Rates	Fixed
I-01	Liquidation Callback Missing Received Amount	Fixed
I-02	Callback Validation Missing for Empty Data	Fixed
I-03	EIP-712 Typehash Parameter Names Mismatch Struct	Fixed
I-04	Rate Changes Blocked During Market Pause	Fixed
I-05	Arbitrary Market Allowance Enables Fund Theft	Fixed

Detailed Findings

High Risk

No issues found.

Medium Risk

M-01 - Liquidation Fee Prioritization Leads to Bad Debt

Severity: Medium Risk

Context:

- AltoBaseMarket.sol#L684-L685

Technical Details:
The liquidation mechanism prioritizes liquidation fees over debt repayment, creating configurations where liquidation fees can consume up to 100% of available collateral regardless of the liquidation LTV.

When liquidation fees are set high relative to the liquidation LTV, liquidators can seize substantial collateral while leaving borrower debt partially unpaid, forcing the protocol to absorb the remaining debt as bad debt. This occurs because the liquidation logic allocates collateral to fees first, then attempts to repay debt with any remaining collateral. In extreme configurations, borrowers can exploit this by maintaining positions that generate profitable liquidations for liquidators while consistently leaving unpaid debt for the protocol to absorb, gradually draining protocol reserves and threatening market solvency.

Impact:
Misaligned liquidation parameters enable systematic bad debt accumulation, potentially leading to market insolvency through repeated exploitation.

Recommendation:
Prioritize debt repayment over liquidation fees and limit the sum of fees and `maxLiquidationLTV` to less than 100%.

Developer Response:
Acknowledged. Admin will be considerate of these edge cases and will apply appropiate limits to `maxLiquidationLtv`.

Low Risk

L-01 - Grace Period Bypassed From Persisting Data

Severity: Low Risk

Context:

- AltoBaseMarket.sol#L676-L677

Technical Details:
The `liquidationConfiguration.isEnabledPriorityLiquidation` check can cause tagged positions to persist in the `liquidablePositions` mapping when they should be removed. When a position is tagged for liquidation but the LTV moves beyond `disablePriorityLiquidationAbovePositionLtv` before liquidation occurs, this check fails and the tagged position remains in the mapping without being deleted.

If the borrower subsequently creates a new liquidatable position, it bypasses the grace period protection because the grace period validation `block.timestamp - liquidablePositions[borrower].timestamp <= gracePeriod` uses the timestamp from the old tagged position rather than treating the new position as requiring a fresh grace period.

Impact:
New borrower positions can be liquidated immediately without proper grace period protection due to stale liquidation timestamps.

Recommendation:
Remove this check and delete the borrower's position from `liquidablePositions` mapping in all cases.

Developer Response:
Fixed at commit `bf65f6a`.

L-02 - Protocol Fee Lost to Zero Address Recipient

Severity: Low Risk

Context:

- AltoBaseMarket.sol#L696-L697

Technical Details:
The `liquidate` function assigns protocol fees to the `feeRecipient` address without validating that it is not the zero address. When `position[feeRecipient].collateralAssets += calc.protocolFee.toUint128()`

executes with `feeRecipient` set to `address(0)`, the protocol fee collateral is credited to the zero address position, making it permanently inaccessible. This represents a loss of protocol revenue since the zero address cannot interact with the contract to claim or utilize the accumulated collateral assets that were intended as protocol fees.

Impact:
Protocol loses fee revenue when the fee recipient is set to the zero address, permanently locking collected fees.

Recommendation:
Before assigning a value to `protocolFee` validate If fee recipient is zero.

Developer Response:
Acknowledged. Admin will be considerate of these edge cases when changing configurations.

L-03 - Fee Recipient Lacks Zero Address Validation

Severity: Low Risk

Context:

- AltoBaseMarket.sol#L271

Technical Details:
The `setBorrowOpeningFee` function lacks validation to ensure the fee recipient is not the zero address when borrow opening fees are set to nonzero values. When a borrow opening fee is configured with a valid percentage but the fee recipient remains unset or is explicitly set to `address(0)`, the protocol will give fee shares to the zero address. These shares become permanently inaccessible since the zero address cannot interact with the protocol, effectively burning the fee revenue that should have been collected by the intended recipient.

Impact:
Fee shares lost, reducing protocol revenue collection.

Recommendation:
Add validation to ensure fee recipient is not zero address when borrow opening fee is nonzero.

Developer Response:
Fixed at `ce57b62`.

L-04 - Borrow Fees Round Down Instead of Up Against Users

Severity: Low Risk

Context:

- AltoBaseMarket.sol#L439
- AltoBaseMarket.sol#L440
- AltoBaseMarket.sol#L446

Technical Details:
The borrow opening fee calculations use rounding methods that favor users instead of the protocol. On line 439, `shares.multiplyWithPrecision` calculates opening fee shares but rounds down when it should round up. On line 440, `convertToAssetsDown` explicitly rounds down when converting fee shares to assets, again favoring the borrower. On line 446, `assets.multiplyWithPrecision` calculates opening fee assets from the total assets but rounds down instead of up.

Since these fees are charged against users as a cost of borrowing, all rounding should favor the protocol by rounding up to ensure users pay the full intended fee amount rather than receiving discounted fees due to precision loss.

Impact:
Protocol collects less fee revenue than intended as users pay reduced opening fees due to rounding in their favor.

Recommendation:
Round up the cited fee calculations since fees are charged against users.

Developer Response:
Fixed at commit `d91fc44`.

L-05 - Calling `setIrm` Will Accrue Interest While Market Paused

Severity: Low Risk

Context:

- AltoBaseMarket.sol#L176-L177

Technical Details:
The `setIrm` function in the BaseMarket contract triggers interest accrual even when the market is paused, violating the protocol's design principle that interest should not accrue during paused states. When the owner calls `setIrm` to update the interest rate model, the function internally calls `_accrueInterest` that calculate and apply interest accrual regardless of the market's paused status.

Impact:
Interest accrues during paused periods contrary to protocol design, creating inconsistent behavior where administrative functions bypass pause restrictions.

Recommendation:
When calling `setIrm`, only accrue interest if the market is not paused.

Developer Response:
Fixed at commit: `c5b7327`.

L-06 - Debt Ceiling Changes Accrue Interest While Paused

Severity: Low Risk

Context:

- AltoMintMarket.sol#L75

Technical Details:
The debt ceiling modification function calls `_accrueInterest` regardless of the market's paused status, violating the protocol's design principle that interest accrual should be suspended during paused periods. When administrators update debt ceiling limits while the market is paused, the function automatically triggers interest calculations and applies accumulated interest that should remain deferred until normal operations resume. This creates an inconsistency where administrative debt ceiling adjustments bypass the pause mechanism's intended restriction on interest accumulation.

Impact:
Interest accrues during paused periods when debt ceilings are modified, creating inconsistent behavior that violates the protocol's pause design.

Recommendation:
Only accrue interest if the market is not paused.

Developer Response:
Fixed at commit `093e7a3`.

L-07 - Taylor Series Approximation Loses Precision at High Rates

Severity: Low Risk

Context:

- FixedPointMath.sol#L41-L42

Technical Details:
The `calculateCompoundInterest` function uses a Taylor series approximation to calculate compound interest. The current implementation becomes increasingly inaccurate as the product of `rate` and `periods` grows larger, particularly when this product exceeds 0.2e18.

In high interest rate environments or when interest accumulates over extended periods, the Taylor series truncation error compounds, causing the function to return systematically lower values than the actual compound interest calculation. This mathematical limitation stems from using only the first few terms of the Taylor series expansion, which provides sufficient accuracy only for smaller input values.

Impact:
Suppliers earn reduced yield in high interest environments or during long accumulation periods due to systematic underestimation of compound interest calculations.

Recommendation:
Add a fourth term to the Taylor series when `rate` multiplied by `periods` exceeds 0.2e18 to improve formula accuracy.

Developer Response:
Fixed at commit `b4adb0f`.

Informational

I-01 - Liquidation Callback Missing Received Amount

Severity: Informational

Context:

- AltoBaseMarket.sol#L702

Technical Details:
The `onLiquidateCallback` function only passes the `f_repayAmount` parameter to the callback contract but omits the actual amount received by the liquidator. Callback contracts can benefit from both the repayment amount and the received collateral amount to accurately perform additional actions such as calculating profit margins, determining optimal liquidation strategies, or executing complex arbitrage operations. The current implementation forces callback contracts to independently calculate the received amount based on liquidation parameters, creating unnecessary complexity and potential for calculation errors.

Impact:
Callback contracts lack complete liquidation information, requiring additional calculations and potentially reducing integration efficiency.

Recommendation:
Add the amount that the liquidator received to the callback parameters.

Developer Response:
Fixed at commit `a7a1a76`.

I-02 - Callback Validation Missing for Empty Data

Severity: Informational

Context:

- AltoBaseMarket.sol#L420-L421

Technical Details:
The `borrow` function lacks validation to ensure that `data` is not empty when the `receiver` is an authorized callback contract. When funds are sent to an authorized callback receiver with empty `data`, the callback call is skipped. When skipped, the funds remain in the receiver contract, effectively trapping the borrowed assets until administrative intervention can recover them.

Impact:
Borrowed funds become stuck in callback contracts when empty data is provided, requiring administrative recovery to retrieve the assets.

Recommendation:
Add validation to ensure data is not empty if the receiver is an authorized callback.

Developer Response:
Fixed at commit `47ab48d`.

I-03 - EIP-712 Typehash Parameter Names Mismatch Struct

Severity: Informational

Context:

- Auth.sol#L16
- AuthUpgradeable.sol#L16

Technical Details:
The EIP-712 typehash definition uses `isAuthorized` as the parameter name in both `Auth` and `AuthUpgradeable` contracts, but the actual `Authorization` struct defines this field as `isAuthorizedStatus`. This naming inconsistency violates EIP-712 conventions where parameter name matching between the typehash string and the struct definition.

The mismatch creates confusion for developers implementing signature verification and reduces code readability by using different names for the same logical field across the typehash and struct definitions.

Impact:
Parameter name inconsistency reduces code readability and does not align with EIP-712 naming conventions for signature verification.

Recommendation:
Change `isAuthorized` to `isAuthorizedStatus` in both typehash definitions to match the struct field name.

Developer Response:
Fixed at commit `9a280eb`.

I-04 - Rate Changes Blocked During Market Pause

Severity: Informational

Context:

- FixedRateIrm.sol#L63

- AdaptiveCurveIrm.sol#90

Technical Details:
The `FixedRateIrm` and `AdaptiveCurveIrm` contracts cannot update borrow rates while the market is paused because the `setBorrowRate` and `setIrmConfig` functions calls the market's `accrueInterest`, which reverts due to a `whenNotPaused` modifier. This creates an operational limitation where administrators cannot adjust interest rates during emergency pause periods, even though changing rates without accruing interest would be safe when no interest accumulation is occurring.

Impact:
Interest rate adjustments cannot be made during paused periods, limiting administrative flexibility for rate management during emergency situations.

Recommendation:
Check and skip `accrueInterest` if the market is paused when calling `setBorrowRate`.

Developer Response:
Fixed at commit `834ffcd`.

I-05 - Arbitrary Market Allowance Enables Fund Theft

Severity: Informational

Context:

- AltoAdapter.sol#L54-L55
- AltoAdapter.sol#L97-L98
- AltoAdapter.sol#L162-L163

Technical Details:
The `AltoAdapter` contract grants unlimited token allowances to unvalidated market addresses using `SafeERC20.forceApprove` with `type(uint256).max`. Since the `market` parameter accepts any address without validation, an attacker can deploy a malicious contract implementing the required interface and pass it as the market parameter. The malicious contract receives unlimited allowance over the adapter's token holdings for both borrow and collateral tokens.

Exploitation follows this pattern:
• Attacker deploys malicious contract with market interface functions
• Attacker calls adapter functions passing malicious contract as `market` parameter
• Adapter grants unlimited allowance to malicious contract via `forceApprove`
• Attacker uses malicious contract to drain any funds existing or later arriving in the adapter

This is only an issue for funds that were either misallocated into the adapter. However a contract giving unlimited allowance that is persistent to arbitrary users would be worth documenting.

Impact:
Attackers can siphon idle funds from the adapter to their own contract.

Recommendation:
It is recommended that this issue be documented.

Developer Response:
Fixed at commit `732cde6`.

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.