SHERLOCK

Security Review For Tenor Labs

Collaborative Audit Prepared For: **Tenor Labs**
Lead Security Expert(s): **0x73696d616f**
xiaoming90
Date Audited: **July 5 - July 7, 2026**

Introduction

Tenor Contracts is a collection of immutable contracts that extend the Morpho stack, primarily Morpho Midnight, Morpho's intent-based fixed-rate lending protocol, alongside Morpho Blue, Morpho Vault V2, and Morpho Oracles. The contracts add stateless callbacks invoked during Morpho Midnight's take(), each encoding an atomic state transition: migrations to and from Morpho Blue and Vault V2, and Midnight-to-Midnight renewals. Together with these migration callbacks, a Migration Ratifier lets users set policies for how their positions can be renewed or migrated on their behalf, and validates each counterparty's offer against the configured parameters. A batch-fill router, exposed through a Bundler3 adapter, can be used to fill a position across multiple offers in one transaction. The repository also includes standalone contracts, including a delayed-liquidation gate and an oracle that implements Morpho's oracle interface and cross-checks its price against a secondary feed for added robustness.

Scope

Repository: tenor-labs/tenor-contracts

Audited Commit: d2cb2ebe3d15ec37aeaf9a5aa19081d53394d8ca

Final Commit: d3756dd834e28c9c57d7191f39482ad218c48df0

Files:

- src/bundler/AuthorizationAdapter.sol
- src/bundler/interfaces/IMidnightAdapter.sol
- src/bundler/interfaces/IMigrationRatifierAdapter.sol
- src/bundler/interfaces/ITenorRouterAdapter.sol
- src/bundler/MidnightAdapterBase.sol
- src/bundler/MigrationRatifierAdapterBase.sol
- src/bundler/TenorAdapter.sol
- src/bundler/TenorRouterAdapterBase.sol
- src/callbacks/BorrowBlueToMidnightCallback.sol
- src/callbacks/BorrowMidnightRenewalCallback.sol
- src/callbacks/BorrowMidnightToBlueCallback.sol
- src/callbacks/interfaces/IBorrowBlueToMidnightCallback.sol
- src/callbacks/interfaces/IBorrowMidnightRenewalCallback.sol
- src/callbacks/interfaces/IBorrowMidnightToBlueCallback.sol
- src/callbacks/interfaces/ILendMidnightRenewalCallback.sol

- src/callbacks/interfaces/ILendMidnightToVaultCallback.sol
- src/callbacks/interfaces/ILendVaultToMidnightCallback.sol
- src/callbacks/interfaces/IMidnightSupplyCollateralCallback.sol
- src/callbacks/interfaces/IMidnightSupplyVaultSharesCallback.sol
- src/callbacks/interfaces/IMidnightWithdrawVaultSharesCallback.sol
- src/callbacks/LendMidnightRenewalCallback.sol
- src/callbacks/LendMidnightToVaultCallback.sol
- src/callbacks/LendVaultToMidnightCallback.sol
- src/callbacks/MidnightSupplyCollateralCallback.sol
- src/callbacks/MidnightSupplyVaultSharesCallback.sol
- src/callbacks/MidnightWithdrawVaultSharesCallback.sol
- src/factories/DelayedLiquidationGateFactory.sol
- src/factories/interfaces/IDelayedLiquidationGateFactory.sol
- src/factories/interfaces/IMidnightAllowlistGateFactory.sol
- src/factories/interfaces/IOracleWithValidationFactory.sol
- src/factories/interfaces/IVaultV2AllowlistGateFactory.sol
- src/factories/MidnightAllowlistGateFactory.sol
- src/factories/OracleWithValidationFactory.sol
- src/factories/VaultV2AllowlistGateFactory.sol
- src/gates/DelayedLiquidationGate.sol
- src/gates/interfaces/IDelayedLiquidationGate.sol
- src/gates/MidnightAllowlistGate.sol
- src/gates/VaultV2AllowlistGate.sol
- src/libraries/CallbackLib.sol
- src/libraries/CollateralTransferLib.sol
- src/libraries/LinearInterpolationLib.sol
- src/libraries/MidnightLib.sol
- src/libraries/PriceLib.sol
- src/libraries/RatePointLib.sol
- src/libraries/RouterLib.sol
- src/libraries/TakeMathLib.sol

- src/libraries/TenorMarketIdLib.sol
- src/oracles/interfaces/IOracleWithValidation.sol
- src/oracles/OracleWithValidation.sol
- src/oracles/README.md
- src/periphery/interfaces/IMidnightVaultExecutor.sol
- src/periphery/MidnightVaultExecutor.sol
- src/periphery/README.md
- src/ratifiers/BaseMigrationRatifier.sol
- src/ratifiers/interfaces/IInterestRatePolicy.sol
- src/ratifiers/interfaces/IMarketMakingPolicy.sol
- src/ratifiers/interfaces/IMigrationRatifier.sol
- src/ratifiers/interfaces/IPausableInterestRatePolicy.sol
- src/ratifiers/interfaces/IRenewalCadence.sol
- src/ratifiers/MigrationRatifier.sol
- src/ratifiers/policies/FourWeekCadence.sol
- src/ratifiers/policies/MarketMakingPolicy.sol
- src/ratifiers/policies/PausableBase.sol
- src/ratifiers/policies/PausableMarketMakingPolicy.sol
- src/ratifiers/policies/PausableStaticRatePolicy.sol
- src/ratifiers/policies/StaticRatePolicy.sol
- src/router/CallbackFeeAdjuster.sol
- src/router/clamps/BorrowBlueToMidnightClamp.sol
- src/router/clamps/BorrowMidnightRenewalClamp.sol
- src/router/clamps/BorrowMidnightToBlueClamp.sol
- src/router/clamps/BuyOfferClamp.sol
- src/router/clamps/LendMidnightRenewalClamp.sol
- src/router/clamps/LendMidnightToVaultClamp.sol
- src/router/clamps/LendVaultToMidnightClamp.sol
- src/router/clamps/SellOfferClamp.sol
- src/router/clamps/SupplyCollateralCallbackClamp.sol
- src/router/clamps/VaultSupplyClamp.sol

- src/router/clamps/VaultWithdrawClamp.sol
- src/router/interfaces/ICallbackFeeAdjuster.sol
- src/router/interfaces/ITakeClamp.sol
- src/router/interfaces/ITenorRouter.sol
- src/router/TenorRouter.sol

Final Commit Hash

d3756dd834e28c9c57d7191f39482ad218c48df0

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	0	1

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue L-1: Executor-gated markets can stall on in-band oracle skew: the documented `MAX_ORACLE_DEVIATION` sizing rule exceeds the executor's liquidation-liveness ceiling [ACKNOWLEDGED]

Source: sherlock-audit issue #81

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

`MidnightVaultExecutor.onLiquidate` funds the repayment strictly from redeeming the seized vault shares in the same transaction. If the redemption proceeds fall short of the repaid amount, the liquidation reverts, and neither the liquidator nor any third party can subsidize the gap, since the check compares against the `redeem()` return value (MidnightVaultExecutor.sol#L161):

```
if (repaidUnits > redeemed) revert RepayExceedsRedeemed();
```

For markets whose collateral is gated VaultV2 shares, this executor is the sole liquidation route in the intended deployment: the vault's shares gate admits only allowlisted receivers, enforced on every share transfer, so `Midnight.liquidate` cannot deliver seized shares to the liquidator directly, and the executor pays only when it is itself the receiver. Midnight's `repay` is also authorization-gated (`midnight/src/Midnight.sol:541`), so executor liquidation is the market's only permissionless risk-reduction path.

Notation used throughout this report (all rates and factors are expressed as plain fractions; in the code they are WAD-scaled, where WAD = 10^18 represents 1.0):

- `LLTV` – the collateral's `lltv` (liquidation loan-to-value), e.g. 0.90 for 90%.
- `c` – the collateral's `liquidationCursor`, the per-collateral parameter that sets the size of the liquidation incentive.
- `maxLif` – the maximum liquidation incentive factor, computed from `LLTV` and `c` by `ConstantsLib.maxLif`.
- `LIF` – the liquidation incentive factor actually applied by `Midnight.liquidate`; in pre-maturity normal mode `LIF = maxLif`.
- `repaidUnits` – the amount of loan-token debt the liquidator repays in the liquidation.
- `redeemed` – the loan tokens the executor actually receives from redeeming the seized shares.
- `P_oracle` – the price per vault share reported by the market oracle, used by Midnight to size the seizure.

- `P_redeem` – the vault's actual redemption rate: the loan tokens `redeem()` pays out per share at execution time.
- `q` – the oracle's overpricing of the shares, measured as a fraction of the oracle price: q = (P_oracle - P_redeem) / P_oracle. For example, q = 5% means the vault pays out 5% less than the oracle's valuation. This is the same measure `MAX_ORACLE_DEVIATION` uses, with the validation feed standing in for the redemption rate.

A liquidation seizes collateral worth (repaidUnits * LIF) at the oracle price, i.e.

sharesSeized = (repaidUnits * LIF) / P_oracle

but the executor realizes those shares at the vault's actual redemption rate, and P_redeem = (1 - q) * P_oracle:

redeemed = sharesSeized * P_redeem = repaidUnits * LIF * (1 - q)

The executor reverts when redeemed < repaidUnits. Substituting the expression above, repaidUnits cancels from both sides, so the condition is independent of the liquidation size:

LIF * (1 - q) < 1 <=> q > 1 - 1/LIF

In pre-maturity normal mode LIF = maxLif, and with (midnight/src/libraries/ConstantsLib.sol)

maxLif = 1 / (1 - c * (1 - LLTV))

the threshold collapses to a simple closed form:

q > 1 - 1/maxLif = c * (1 - LLTV)

The executor's tolerance to oracle overpricing is therefore c * (1 - LLTV): the liquidation bonus expressed as a fraction of the oracle price. Every liquidation, of any size, reverts as soon as q exceeds it.

`OracleWithValidation.price()` bounds exactly this measure: it reverts when the primary-vs-validation deviation, as a fraction of the primary price, exceeds the immutable `MAX_ORACLE_DEVIATION`, so with the validation feed as the proxy for the redemption rate, the market keeps operating for all q <= `MAX_ORACLE_DEVIATION`. (Caveat: this cap works only while the validation check is active and the validation feed stays close to the redemption rate. If the check is paused, or the validation call fails while `REVERT_ON_VALIDATION_ORACLE_FAILURE = false`, `price()` returns the primary price unchecked and nothing caps q.)

The in-scope sizing guidance (`tenor-contracts/src/oracles/README.md`, section "Setting MAX_ORACLE_DEVIATION") only instructs deployers to keep this threshold below the bad-debt bound, which is expressed in the same measure as q:

> Below `(1 - LLTV * maxLif)`, where `maxLif = 1 / (1 - liquidationCursor * (1 - LLTV))` is the per-collateral maximum liquidation incentive factor. [...] Set the deviation below `(1 - LLTV * maxLif)` per collateral, with headroom for natural oracle drift.

In the notation of this report the README ceiling is

1 - LLTV * maxLif = ((1 - c) * (1 - LLTV)) / (1 - c * (1 - LLTV))

The README ceiling drops to or below the executor ceiling c * (1 - LLTV) only when

((1 - c) * (1 - LLTV)) / (1 - c * (1 - LLTV)) <= c * (1 - LLTV) <=> c >= 1 / (1 + sqrt(LLTV))

a threshold always slightly above 0.5 (about 0.513 at LLTV = 0.9).

For any smaller cursor there exists a band of deviations

q in (c * (1 - LLTV), 1 - LLTV * maxLif]

that simultaneously:

- passes `OracleWithValidation.price()` (in-band, so no `ExcessiveOracleDeviation` revert; the market operates normally);
- contains liquidatable positions that carry no bad debt by Midnight's accounting (LLTV * maxLif < 1 leaves a gap between the liquidation and bad-debt thresholds); yet
- makes every liquidation through the executor revert with `RepayExceedsRedeemed`.

Neither the README nor any other in-scope document states the joint constraint `MAX_ORACLE_DEVIATION` < c * (1 - LLTV) for executor-routed markets.

Post-maturity ramp: a bounded dead window remains even with correct sizing

The analysis above uses pre-maturity normal mode, where LIF = maxLif. After maturity, Midnight ramps the LIF linearly from 1 at maturity up to maxLif over `TIME_TO_MAX_LIF` (60 minutes) (`midnight/src/Midnight.sol:685-687`):

Midnight.sol#L685

```
uint256 lif = postMaturityMode
    ? UtilsLib.min(
        _maxLif,
        WAD + (_maxLif - WAD) * (block.timestamp - market.maturity)
            / TIME_TO_MAX_LIF
    )
    : _maxLif;
```

In settlement mode the executor's skew tolerance at time t after maturity is therefore 1 - 1/LIF(t), which starts at zero and only reaches c * (1 - LLTV) at the end of the ramp. A skew of q blocks executor liquidations in post-maturity settlement mode until LIF(t) reaches 1 / (1 - q), i.e. for the first

t_dead = 60 * q / ((1 - q) * (maxLif - 1)) minutes

after maturity. In the example market below, a modest q = 1% skew delays post-maturity settlement through the executor by about 24 minutes.

The ramp governs settlement mode only: an actually-unhealthy position remains liquidatable at any time through the health-based normal mode (`postMaturityMode = false`, requiring originalDebt > maxDebt) at the full maxLif with no ramp, so the window only delays permissionless settlement of still-healthy positions.

Sizing `MAX_ORACLE_DEVIATION` < c * (1 - LLTV) keeps the window under the full 60 minutes but does not remove it. The executor NatSpec says "the first seconds of the post-maturity incentive ramp"; the window scales with q and can span most of the hour, so settlement automation must not assume execution immediately at maturity.

Numeric example

Market: LLTV = 0.90, c = 0.25, gated VaultV2-share collateral, `OracleWithValidation` sized per the README.

- maxLif = 1 / (1 - 0.25 * 0.10) = 1.02564
- Executor liveness ceiling = c * (1 - LLTV) = 2.50%
- README's recommended ceiling = 1 - 0.90 * 1.02564 = 7.69%
- Stall band = (c * (1 - LLTV), 1 - LLTV * maxLif] = (2.50%, 7.69%], a 3.08x gap

Now suppose the vault takes a loss and the primary oracle lags: the vault now pays out 5% less than the oracle's valuation, i.e. q = 5%. This is inside the band the README allows, so `price()` does not revert and the market keeps operating normally.

A liquidator repays 10,000 loan units against an unhealthy borrower:

- Shares seized, valued at the oracle price: 10,000 * 1.02564 = 10,256.41 units (exact, since maxLif = 40/39)
- Actual redemption proceeds: 10,256.41 * 0.95 = 9,743.59 units
- Check: repaidUnits (10,000) > redeemed (9,743.59), so the call reverts with `RepayExceedsRedeemed`

The shortfall of about 256 units scales linearly with the repay size, so partial liquidations revert identically. The position stays open and the freeze lasts until the oracle-vs-redemption skew falls back under 2.50%.

Impact

A team that follows the project's own published sizing rule launches executor-gated markets with a built-in liquidation-freeze band. When a skew event lands in that band (a vault loss with a lagging oracle, a partial oracle correction, or a structural oracle-above-redemption gap such as an exit fee), `price()` keeps returning normally and the market keeps operating, but every liquidation attempt through the executor reverts with `RepayExceedsRedeemed`. In the frozen configuration (nobody else allowlisted, gate ownership renounced) there is no fallback, and liquidations stay dead for as long as the skew persists.

Time alone does not create the loss (debt is a fixed amount of units due at maturity), but if the collateral's redemption value keeps falling, or the oracle corrects only partially, while liquidations are frozen, positions cross the point where seizing all collateral no longer covers the debt; the eventual liquidation then books the shortfall as bad debt and socializes it across the market's lenders via the loss factor.

Code Snippet

MidnightVaultExecutor.sol#L161

Tool Used

Manual Review

Recommendation

1. Document the joint constraint that for markets whose collateral is liquidated through `MidnightVaultExecutor`, the binding ceiling is

   `MAX_ORACLE_DEVIATION` < 1 - 1/maxLif = c * (1 - LLTV)

   with the same headroom left for honest oracle drift.

   Note the residual post-maturity window: an in-band skew q still delays executor settlement-mode liquidations for 60 * q / ((1 - q) * (maxLif - 1)) minutes after maturity.

2. Prefer c >= 1 / (1 + sqrt(LLTV)) for executor-routed markets (about 0.513 at LLTV = 0.9, 0.528 at LLTV = 0.8, 0.586 at LLTV = 0.5). When c is at or above it, any README-compliant `MAX_ORACLE_DEVIATION` is automatically executor-safe. For lower cursors the joint constraint can be unsatisfiable: a c * (1 - LLTV) of a few percent may leave no value both above honest feed spread and below the executor ceiling, and the sound fix is then raising the cursor, not squeezing the threshold.
3. Enforce the constraint at ratification, not in the factory (`OracleWithValidationFactory` receives no market parameters), and monitor oracle-vs-redemption divergence in production: tightening the threshold only converts the silent stall into a `price()` halt, so it must still stay above honest feed drift.

Discussion

hboisselle

Ack and accepted. We'll never use validation oracles for vault collaterals. Nonetheless reworded the MidnightVaultExecutor natspec to be a little bit to be more precise about the post-maturity liquidation incentive ramp tenor-contracts PR #24

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.