

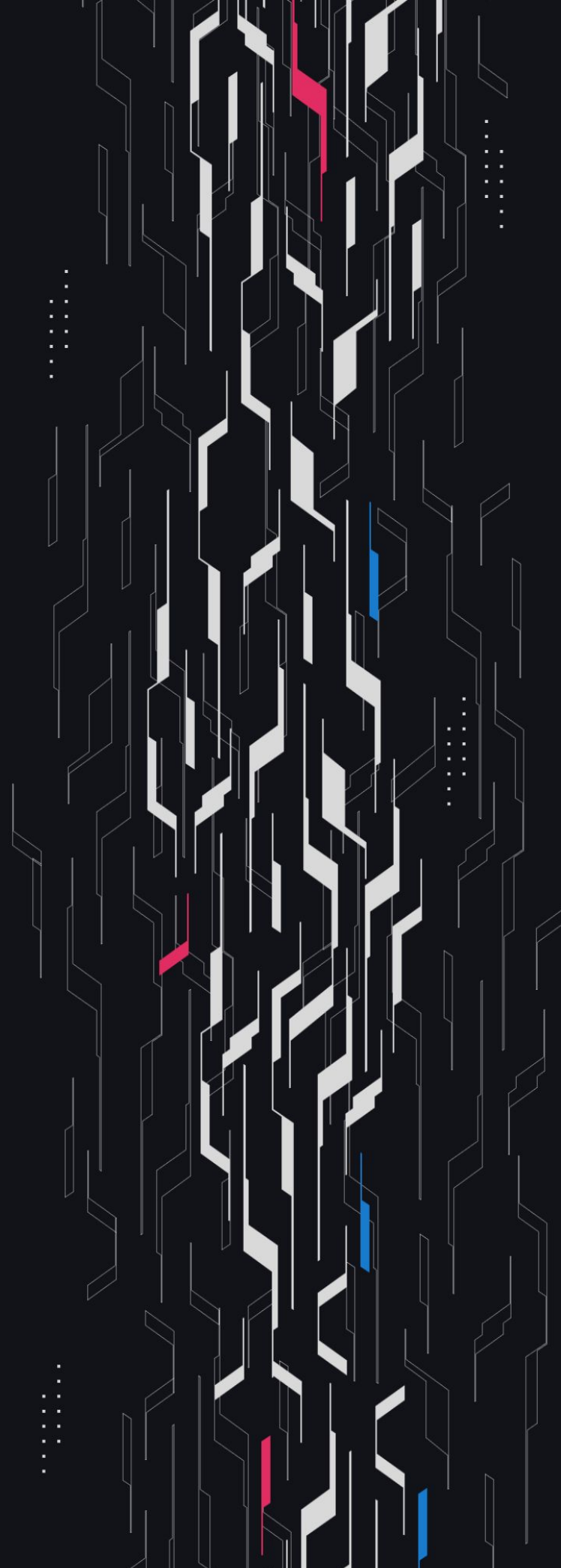
GA GUARDIAN

Tenor

Lending Protocol

Security Assessment

May 12th, 2025



Summary

Audit Firm Guardian

Prepared By Roberto Reigada, Osman Ozdemir, Wafflemakr,

Cosine, Nicholas Chew, Michael Lett

Client Firm Tenor

Final Report Date May 12, 2025

Audit Summary

Tenor engaged Guardian to review the security of their Tenor protocol contracts. From the 7th of April to the 21st of April, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking & Security Recommendation

Given the lack of critical issues detected and minimal code changes following the main review, Guardian assigns a Confidence Ranking of 4 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.



Verify the authenticity of this report on Guardian's GitHub: <https://github.com/guardianaudits>



Code coverage & PoC test suite: <https://github.com/GuardianOrg/tenor-amm-fuzz>

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	<p>Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design.</p> <p>Recommendation: Code is highly secure at time of audit. Low risk of latent critical issues.</p>	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	<p>Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed.</p> <p>Recommendation: Suitable for deployment after remediations; consider periodic review with changes.</p>	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	<p>Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases.</p> <p>Recommendation: Address issues thoroughly and consider a targeted follow-up audit depending on code changes.</p>	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	<p>Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues.</p> <p>Recommendation: Post-audit development and a second audit cycle are strongly advised.</p>	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	<p>Code has systemic issues. Multiple High/Critical findings (≥ 5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured.</p> <p>Recommendation: Halt deployment and seek a comprehensive re-audit after substantial refactoring.</p>	≥ 5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Information

Project Overview 5

Audit Scope & Methodology 6

Smart Contract Risk Assessment

Invariants Assessed 9

Findings & Resolutions 15

Addendum

Disclaimer 50

About Guardian Audits 51

Project Overview

Project Summary

Project Name	Tenor
Language	Solidity
Codebase	https://github.com/Shippooor-Labs/tenor-amm/
Commit(s)	Initial commit(s): e165d6c7c67212447738e44d6b6279f0b383ff0b Final commit: 843c87a8256cdcc4707827f0fe8db7c0773f8fc4

Audit Summary

Delivery Date	May 12, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● High	1	0	0	0	0	1
● Medium	4	0	0	1	0	3
● Low	27	0	0	17	0	10
● Info	0	0	0	0	0	0

Audit Scope & Methodology

Initial review repository:

Review commit: [e165d6c7c67212447738e44d6b6279f0b383ff0b](#)

Fix review repository:

Review commit: [843c87a8256cdcc4707827f0fe8db7c0773f8fc4](#)

Scope and details:

contract,source,total,comment

./src/BasePoolManager.sol,233,392,90

./src/Extload.sol,22,34,10

./src/LiquidityToken.sol,37,57,13

./src/PoolAccounting.sol,66,153,70

./src/PoolManager.sol,55,80,14

./src/PoolManagerHook.sol,168,264,60

./src/PoolManagerRoles.sol,160,265,39

./src/types/Errors.sol,36,38,1

./src/types/ExchangeRate.sol,57,101,23

./src/types/PoolId.sol,8,17,6

./src/types/PoolKey.sol,11,25,11

./src/types/Seconds.sol,14,27,8

./src/types/TickBitmap.sol,49,83,20

./src/libraries/ERC4626Utils.sol,9,20,8

./src/libraries/HookPermissions.sol,11,15,7

./src/libraries/HookPoolKey.sol,57,134,61

./src/libraries/LiquidityTokenId.sol,18,35,13

./src/libraries/LiquidityUtils.sol,31,74,29

./src/libraries/PoolActions.sol,415,681,157

./src/libraries/PoolManagerStateReader.sol,474,760,168

./src/libraries/SafeCast.sol,7,14,6

./src/libraries/SwapConversion.sol,39,68,19

./src/libraries/SwapUtils.sol,111,185,59

./src/storage/PoolManagerStorage.sol,100,126,27

./src/storage/StorageLayout.sol,24,32,3

./src/periphery/PoolManagerView.sol,50,145,79

./src/factories/PoolManagerFactory.sol,13,22,6

./src/factories/PoolManagerHookFactory.sol,15,25,6

source count: {total: 3872, source: 2290, comment: 1013, single: 875, block: 138, mixed: 30, empty: 599, todo: 0, blockEmpty: 0, commentToSourceRatio: 0.4423580786026201}

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● Medium	● Low	● Low

Impact

- High** Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
- Medium** A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
- Low** Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

- High** The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
- Medium** An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
- Low** Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian’s review of Tenor, fuzz-testing was performed on the protocol’s main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
G-01	addLiquidity shouldn't fail for valid calls			N/A	10M+
G-02	withdrawLiquidity shouldn't fail for valid calls			N/A	10M+
G-03 (G-01R)	swap shouldn't fail for valid calls				10M+
G-04 (G-02R)	withdrawLimitOrder shouldn't fail for valid calls				10M+
G-05 (G-03R)	simulateSwap shouldn't fail for valid calls				10M+
G-06 (G-04R)	claimFees shouldn't fail for valid calls				10M+
G-07 (G-05R)	initialize shouldn't fail for valid calls				10M+
G-08 (G-06R)	setFeeShares shouldn't fail for valid calls				10M+
G-09	simulateAddLiquidity shouldn't fail for valid calls			N/A	10M+
G-10 (G-07R)	The PoolManagerHook contract should never hold any funds as everything should be in Uniswap.				10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
G-11 (G-08R)	After initialising a PoolManagerHook pool no swaps should happen on uniswap and no liquidity should be provided therefore the related params should not change.				10M+
PG-01	A pool cannot be initialized with a poolKey that already exists				10M+
PG-02	The pool cannot be initialized if the fixedToken does not meet ERC20 or ERC4626 standards				10M+
PG-03	The pool cannot be initialized if the loanToken does not meet ERC20 or ERC4626 standards				10M+
PG-04	When using ERC4626 for the loan token, the underlying token decimals should be < loan token decimals + 18				10M+
PG-05	The pool's maturity must be higher than the current timestamp during initialization				10M+
PG-06	The pool's maturity must be no later than 500 days from the initialization timestamp				10M+
PG-07	(Hook only) The pool's maturity must be rounded to the nearest hour				10M+
PG-08	(Hook only) A unique Uniswap pool is always initialized for each individual Tenor pool				10M+
PG-09	(Hook only) Each Uniswap pool maps to a single Tenor pool, and vice versa				10M+
PG-10	Pool initialization is permissionless if the initializer address is set to zero				10M+
PG-11	Only initializer may initialize pools if the initializer address is non-zero				10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
PI-01	Operations in one pool must not influence the state of any other pool	✓	✓	✓	10M+
PI-02	Actions within a pool should only alter the balances of the loanToken and fixedToken for that specific pool as maintained by the pool manager contract	✓	✓	✓	10M+
PA-01	sum of limit orders, account balances and fees should equal to total minted for each token	✓	✓	✓	10M+
PA-02	sum of shares from all actors should equal to total shares for each liquidity token	✓	✓	✓	10M+
PA-03	After withdrawing a limit order, the 6909 balance change should reflect the change in shares burned	✓	✓	✓	10M+
PA-04	In a withdraw limit order transaction, the number of tokens removed from the pool manager contract must not exceed the number of tokens present in the pool before the transaction	✓	✓	✓	10M+
PA-05	Adding and then immediately withdrawing a limit order must result in no positive change to the user's balance. The user balance can decrease marginally due to rounding	✓	✓	✓	10M+
PA-06	In a withdraw liquidity transaction, the number of tokens removed from the pool manager contract must not exceed the number of tokens present in the pool before the transaction	✓	✓	N/A	10M+
PA-07	In a withdraw limit order transaction, the number of tokens removed from the pool manager contract must not exceed the number of tokens present in the pool before the transaction	✓	✓	N/A	10M+



















Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
OB-01	Bid <= Ask	✓	✓	✓	10M+
OB-02	bidBitmap should match fixed token balances in unfilled and partially filled limit orders	✓	✓	✓	10M+
OB-03	askBitmap should match loan token balances in unfilled and partially filled limit orders	✓	✓	✓	10M+
OB-04	A limit bid (using Fixed Tokens) cannot be set at a tick above the current best ask	✓	✓	✓	10M+
OB-05	A limit ask (using Loan Tokens) cannot be set at a tick below the best bid	✓	✓	✓	10M+
OB-06	Adding Fixed Tokens above the best ask tick is not allowed	✓	✓	✓	10M+
OB-07	Adding Loan Tokens below the best bid tick is not allowed	✓	✓	✓	10M+
OB-08	The buying power of the LP should not make a stepwise jump after calling addLiquidity.	✓	✓	✓	10M+
SWP-01	After a swap, the number of limit order shares held by LPs must remain unchanged	✓	✗	✗	10M+
SWP-02	After a swap, the set of active ticks in the bid and ask bitmaps should never increase, it can stay the same or decrease	✓	✗	✓	10M+
SWP-03	For a swap with fixedTokenForLoanToken == True, after swap best Bid/Ask >= Before swap best Bid/Ask	✓	✓	✓	10M+
SWP-04	For a swap with fixedTokenForLoanToken == False, after swap best Bid/Ask <= Before swap best Bid/Ask	✓	✓	✓	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
SWP-05	For swaps, the pool manager contract's output amount (amountOut) must not exceed the total quantity of tokenOut in the pool before the swap	✓	✓	✓	10M+
SWP-06	A swap must not result in a user's trade implied rate below 0%	✓	✗	✗	10M+
SWP-07	A swap must not result in a trade implied rate above: $\text{maxTick} * \text{bpsPerTick} + \text{bpsPerTick}$	✓	✓	✓	10M+
SWP-08	The exchange rate between Underlying Tokens (computed as $\text{Loan Tokens} * \text{loanToUnderlyingExchangeRate}$) and Fixed Tokens must always exceed 1	✓	✓	✓	10M+
SWP-09	For a swap from Loan Tokens to Fixed Tokens, the user's trade implied interest rate at each tick must be lower than the spot tick interest rate (to account for fees). In other words, lenders are paying fees	✓	N/A	✓	10M+
SWP-10	For a swap from Fixed Tokens to Loan Tokens, the implied interest rate must be higher than the spot tick interest rate. In other words, borrowers are paying fees	✓	N/A	✓	10M+
SWP-11	If a user swaps loan tokens for fixed tokens (or vice versa) and then immediately swaps back their entire position, the final balance must be lower than the initial balance due to fees on both swaps	✓	N/A	✓	10M+
SWP-12	When swapping against an unfilled limit order tick: There must be no remaining partial limit order at the same tick. If a partial order exists, it must be completely filled before interacting with the unfilled limit order	✓	N/A	✓	10M+
FEE-01	During all swaps, the PoolManager's FixedTokenFees and LoanTokenFees must never decrease	✓	✓	✓	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
FEE-02	These fees should only increase during swaps; adding or removing liuqidity or limit orders should not alter them				10M+
FEE-03	For any swap, only one fee type (currencyIn) should increase; If fixedFeesChange > 0, then loanFeesChange == 0; If loanFeesChange > 0, then fixedFeesChange == 0				10M+
FEE-04	Fees should not accrue after maturity since swapping is not possible after maturity				10M+
FEE-05	If LimitFeeShare is set to 0 for a given pool, then the pool manager should never accumulate fees for that pool				10M+
FEE-06	The total amount of loanTokenFees and fixedTokenFees should never be larger than the amount held by the pool manager contract				10M+
FEE-07	Fees can only decrease when someone calls the claimOwnerFees or claimProtocolFees function				10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Uniswap V4 Swaps Revert Due To Invalid Slippage	Logical Error	● High	Resolved
M-01	Spot Tick “Add-Then-Remove” Bypass Enables Near-Free Swaps	Warning	● Medium	Acknowledged
M-02	Incorrect assertCompatibilityWithUniswap - beforeInitialize	Validation	● Medium	Resolved
M-03	Pools Can Be Initialized Before Protocol Fee Is Set	Logical Error	● Medium	Resolved
M-04	Slippage Risk For Swap Limit Orders	Validation	● Medium	Resolved
L-01	Exchange Rate Does Not Always Round In LP Favor	Rounding	● Low	Acknowledged
L-02	Order-Filling Priority Can Be Gamed Easily	Warning	● Low	Acknowledged
L-03	Missing beforeHook Overwrite In HookPermissions	Warning	● Low	Resolved
L-04	Incorrect NatSpec Comment In Swap Function	Documentation	● Low	Resolved
L-05	Potential High Gas On Market Orders	Warning	● Low	Acknowledged
L-06	Missing Multicall In PoolManager Contract	Warning	● Low	Acknowledged
L-07	Unused Swap Callback	Warning	● Low	Acknowledged
L-08	Frontrun Vault Update For Risk-Free Profit	MEV	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-09	Lack Of Incentives To Lend In Tenor	Logical Error	● Low	Acknowledged
L-10	Incorrect Fee Accounting During exactIn Swaps	Logical Error	● Low	Resolved
L-11	Incorrect Errors Are Used In PoolAccounting	Informational	● Low	Resolved
L-12	No Callback Support For Liquidity Operations	Informational	● Low	Acknowledged
L-13	Misleading Event Emission	Informational	● Low	Acknowledged
L-14	Withdrawing 0 Shares Is Possible	Informational	● Low	Resolved
L-15	ERC6909 Approvals And Operator Ignored	Access Control	● Low	Acknowledged
L-16	loanToAsset Does Not Account For Slippage/Fees	Warning	● Low	Acknowledged
L-17	Unused Error In PoolActions	Informational	● Low	Resolved
L-18	Missing Max bpsPerTick Check In Init Flow	Validation	● Low	Acknowledged
L-19	Functions Naming Convention	Best Practices	● Low	Resolved
L-20	Mismatched Comments For Fee And Tick Spacing	Documentation	● Low	Resolved
L-21	External Incentives Are Lost	Rewards	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-22	Typos In Protocol Documentation	Informational	● Low	Resolved
L-23	Uniswap V4 DoS During Callback	DoS	● Low	Acknowledged
L-24	Rethink Uniswap Integration	Logical Error	● Low	Acknowledged
L-25	Lack Of Incentives To Provide Liquidity	Validation	● Low	Acknowledged
L-26	Failed Invariants For Edge Cases	Unexpected Behavior	● Low	Resolved
L-27	Liquidity Provision Is Capital Inefficient	Rewards	● Low	Acknowledged

H-01 | Uniswap V4 Swaps Revert Due To Invalid Slippage

Category	Severity	Location	Status
Logical Error	● High	BasePoolManager.sol	Resolved

Description

When performing a swap through the Uniswap v4 Router, the `sqrtPriceLimitX96` parameter is automatically set to either `TickMath.MIN_SQRT_PRICE + 1` or `TickMath.MAX_SQRT_PRICE - 1` ([V4Router.sol#164](#))

This value is passed into `SwapConversion.toTenorParams`, where it is used to compute `slippageLimit` for the Tenor protocol:

```
tenorSwapParams.slippageLimit = uniswapParams.sqrtPriceLimitX96 - MIN_PRICE_LIMIT
```

As a result, `slippageLimit` ends up being either 0 or a large constant near the full sqrt price range. This causes Tenor’s internal `_checkMaxSlippageReached` to always revert in certain swap directions (e.g., exact input swaps from `currency1` to `currency0`).

In the opposite direction, swaps do not revert but `slippageLimit` becomes meaningless. Since users executing swaps through the Uniswap Router have no way to customize slippage settings, Uniswap-based swap routes into Tenor are effectively broken.

Recommendation

Consider using the `hookdata` parameter to provide slippage values instead of deriving them from Uniswap’s `sqrtPriceLimitX96`.

Resolution

Tenor Team: The issue was resolved in commit [1da33c4](#).

M-01 | Spot Tick “Add-Then-Remove” Bypass Enables Near-Free Swaps

Category	Severity	Location	Status
Warning	● Medium	Global	Acknowledged

Description

Tenor enforces that liquidity providers at the spot tick must deposit and withdraw tokens proportionally so they cannot swap without incurring normal swap fees. This implementation is also clearly defined in these [docs](#).

However, users can front-run a large swap at the spot tick to perform a free or near-free token exchange:

1. The spot tick currently holds X loan and X fixed, implying a nominal 1:1 rate.
2. The user sees a pending transaction on the mempool that will trade a large chunk of tokens at the spot tick, shifting its ratio significantly (e.g., from 1:1 to 1:1.66).
3. Before the large swap transaction is executed, the attacker adds liquidity to the spot tick proportionally, depositing (X, X) of (loan, fixed), doubling the total to (2X, 2X).
4. The big swap then executes, consuming tokens from the spot tick, e.g., leaving (1.5X loan, 2.5X fixed).
5. Right after that trade, the attacker withdraws their liquidity. Because they deposited 50% of the tick’s total before the swap, they now remove 50% of the updated loan/fixed pool. If the trade shifted the ratio to 1:1.66, the user’s portion is no longer 1:1 but includes a bigger portion of one token, netting a no-fee or arbitrage swap.

Despite Tenor’s rule that spot tick deposits must be “proportional,” the post-swap ratio changes within the spot tick, so the attacker’s withdraw (still the same share fraction) entitles them to a newly skewed ratio. This yields a near-free token exchange, front-running the big swap.

They effectively circumvent normal swap paths/fees because the code sees them as legitimate LP actions. Finally it is worth mentioning that in this case, the liquidity provider also managed to take a portion of the liquidity fee of the previous swap.

Recommendation

Consider enforcing a minimum lock period once a user adds liquidity to the spot tick, before they can withdraw. This prevents rapid deposit–withdraw cycles around a large trade.

Resolution

Tenor Team: Acknowledged.

M-02 | Incorrect assertCompatibilityWithUniswap - beforeInitialize

Category	Severity	Location	Status
Validation	● Medium	PoolManagerHook.sol	Resolved

Description

Within Tenor’s SwapConversion library, the toTenorParams and toUniswapParams functions, along with toTenorKey and toUniswapKey in HookPoolKey, are intended to be symmetrical. That is, converting a TenorPoolKey → UniswapPoolKey → back to TenorPoolKey should yield the same or revert if incompatible.

However, this symmetry breaks under certain conditions:

1. If bpsPerTick > 127: The “7-bit” range in the Uniswap key layout allows [1..127]. If the original Tenor bpsPerTick is outside that range, toUniswapKey becomes is truncated, while converting back with toTenorKey quietly folds it into some permissible default.
2. If maturity timestamp not a multiple of 1 hour.

assertCompatibilityWithUniswap is called in two places:

- initialize: receives the raw user-provided TenorPoolKey, so it properly checks if (bpsPerTick <= 127, hour-aligned maturity, etc.). It will revert right away if an invalid value is provided.
- beforeInitialize: receives uniswapKey.toTenorKey(). But if the Uniswap key was already invalid or out of range, toTenorKey quietly produces a “valid” but a “truncated” Tenor key. So the check never fails, the code can’t detect that the original bits were out of range.

The main impact is that pools that are created through the UniswapV4 PoolManager contract directly may be configured with incorrect parameters and the deployer would not be able to detect it. For example, deploying the following Pool through UniswapV4 PoolManager:

```
PoolKey memory poolKey = PoolKey({loanToken: address(contract_ERC4626_loanToken), fixedToken:
address(contract_ERC20_fixedToken), maxTick: 16, bpsPerTick: 200, maturity:
uint32(Seconds.unwrap(maturity)), loanTokenIsERC4626: true});
```

Would instead setup the following underlying Tenor pool:

```
emit InitializePool(id: 0x62fc492d78e5ebbca14c6e5dc68be27b2305379d9a0027cbdc135862f05e233e, key:
PoolKey({ loanToken: 0x8c15701ab95A8760403009414B830C7df9E4a4dc, fixedToken:
0xEdB52d394123f41118e42Fb0f19008E8022529e1, maxTick: 17, <----- bpsPerTick: 72, <-----
maturity: 1776171600 [1.776e9], loanTokenIsERC4626: true}))
```

Recommendation

Update the beforeInitialize function to correct this issue or otherwise, consider documenting the supported ranges and informing the users of this limitation when deploying directly through the UniswapV4 PoolManager.

Resolution

Tenor Team: The issue was resolved in commit [8ab2e7a](#).

M-03 | Pools Can Be Initialized Before Protocol Fee Is Set

Category	Severity	Location	Status
Logical Error	● Medium	BasePoolManager.sol	Resolved

Description

A portion of the pool fees goes to Tenor as protocol fees, while the remaining portion goes to pool owners as owner fees. These shares are determined based on `protocolShareOfFeeShares`, which is set by Tenor as a global configuration value.

The `protocolShareOfFeeShares` for a pool is set to the global configuration value at the time of pool initialization and cannot be changed afterward.

The issue is that the global `protocolShareOfFeeShares` is neither set at the time of deployment nor enforced with a minimum value, and must be set by Tenor after deployment.

Other protocols integrating with Tenor will be the owners and will have their own `PoolManager` instances, which they can create via factory contracts.

Since there is a time gap between deployment and configuration, these protocols can initialize all major pools immediately after deployment, before the `protocolShareOfFeeShares` is set by Tenor, and receive all the fees themselves.

Recommendation

Set a base value for `protocolShareOfFeeShares` at the time of deployment to prevent revenue loss to other protocols, while retaining the ability to modify it later if necessary.

Resolution

Tenor Team: The issue was resolved in commit [bbf8a20](#).

M-04 | Slippage Risk For Swap Limit Orders

Category	Severity	Location	Status
Validation	● Medium	BasePoolManager.sol: 305-31	Resolved

Description

Market orders and slippage orders are executed with the same function. The implemented slippage check is based on the filled amount, but for limit orders zero size is filled.

Therefore if a user expects a pure limit order the user needs to set the slippage check to 0 so that it does not revert. It is also possible to perform a partial limit and partial market order and here this slippage check can be imprecise.

For example:

- Eve provided most of the liquidity to the pool
- Alice wants to perform a big swap through multiple ticks to clear all of the available liquidity and create a limit order at the last one to incentivize other users to fill it soon (as she expects a good part of the order to be created as limit order she needs to reduce the expected returned filled amount for the slippage check by that)
- Eve front runs the call and redistributes the liquidity so that Alice swaps as much as possible under the worst condition right before the limit tick
- Alice received less than expected but was not able to prevent that as the slippage check takes the unfilled limit order part into account

Recommendation

Consider adjusting the slippage check to only account for the market order part.

Resolution

Tenor Team: The issue was resolved in commit [21a07df](#).

L-01 | Exchange Rate Does Not Always Round In LP Favor

Category	Severity	Location	Status
Rounding	● Low	ExchangeRate.sol, ERC4626Utils.sol	Acknowledged

Description

The protocol is designed to favor liquidity providers (LPs) in swap execution, as seen in `calculateAmountIn` and `calculateAmountOut`, which intentionally round amounts to the LP's advantage.

However, in `ERC4626Utils.sol:loanToAsset` and `ExchangeRate.sol:toExchangeRate`, the calculated exchange rate always rounds down, regardless of swap direction or type. In certain cases, this benefits the swapper instead of the LP.

Recommendation

When calculating the exchange rate at `loanToAsset` and `toExchangeRate`, the the rounding direction should be aligned with the swap context to ensure LP-favorable execution:

Round down for:

- `fixedForLoan` + exact output
- `loanForFixed` + exact input

Round up for:

- `fixedForLoan` + exact input
- `loanForFixed` + exact output

Resolution

Tenor Team: Acknowledged. Exchange rates always need to be rounded the same way to ensure consistency across the protocol.

L-02 | Order-Filling Priority Can Be Gamed Easily

Category	Severity	Location	Status
Warning	● Low	PoolActions.sol: 489	Acknowledged

Description

In the current design, Tenor processes swaps by first filling partially filled limit orders, then unfilled limit orders and finally the pool’s spot liquidity. This ordering means that once a limit order is moved from “unfilled” to “partial” it gains top priority in subsequent swaps.

By performing a minimal self swap of size 1 or another small amount against a previously unfilled limit order, an attacker can convert that limit order into a partially filled order, thereby elevating it in the queue for subsequent trades at that tick.

This small fill transforms the order from “unfilled” to “partial,” causing it to be matched first in any future large swap arriving at that tick.

Recommendation

Consider enforcing the same priority among unfilled and partially filled limit orders or add constraints preventing trivial minimal fills that move a large full order into the partial queue.

Resolution

Tenor Team: Acknowledged. A limit order user can only do this if there is no outstanding partially filled limit orders in the same tick and at the spot tick.

L-03 | Missing beforeHook Overwrite In HookPermissions

Category	Severity	Location	Status
Warning	● Low	HookPermissions.sol	Resolved

Description

Currently, Tenor’s HookPermissions for beforeDonate is set to false. As a result, whenever users call donate in a Uniswap V4 pool with PoolManagerHook as the hook, the call will fail with a NoLiquidityToReceiveFees error logged directly from the Uniswap V4 pool instead of reverting with a HookNotImplemented error.

Because Tenor already follows the pattern of turning hooks on but reverting for standard beforeAddLiquidity and beforeRemoveLiquidity flows, the same approach could be adopted for beforeDonate.

Recommendation

Consider setting beforeDonate = true in HookPermissions, so the Uniswap V4 manager calls beforeDonate(...) reverting with the actual HookNotImplemented error.

Resolution

Tenor Team: The issue was resolved in commit [a6ca005](#).

L-04 | Incorrect NatSpec Comment In Swap Function

Category	Severity	Location	Status
Documentation	● Low	IPoolManager.sol	Resolved

Description

Tenor’s IPoolManager.swap function includes a slippageLimit parameter documented as follows:

```
/// @param slippageLimit Sets the slippage limit for the swap. /// Specified as
maximum amount in if swap is exact out. /// Max amount out if swap is exact in
```

This part is incorrect and does not reflect the current implementation:

```
/// Max amount out if swap is exact in
```

It should be instead:

```
/// Min amount out if swap is exact in
```

On the other hand, the NatSpec comments for the swap function in IPoolManager contract omit documentation for the loanForFixed parameter and redundantly document the limitTick parameter twice.

Recommendation

Update the comments as suggested in the IPoolManager interface.

Resolution

Tenor Team: The issue was resolved in commit [cc5972c](#).

L-05 | Potential High Gas On Market Orders

Category	Severity	Location	Status
Warning	● Low	PoolActions.sol: 477	Acknowledged

Description

In Tenor’s PoolManager.swap logic, each incoming market swap iterates over ticks in ascending or descending order, potentially hitting all ticks in the pool.

Since Tenor can support up to 111 ticks (from 1 to 111), a large order with low liquidity might traverse most or all ticks to fill the user’s requested amount. In each tick, the code proceeds through partial limit orders, unfilled limit orders and finally liquidity.

A stress test has shown that, when the user triggers many small partial fills across many ticks, the gas usage can approach or exceed 22 million gas, near the block limit on certain networks. This can cause out-of-gas reverts or extremely expensive calls.

Recommendation

Consider bootstrapping key ticks with sufficient liquidity so fewer ticks must be crossed for typical trades. On the other hand, for lightly used deployments, keep maxTick smaller than 111 to reduce iteration count.

Resolution

Tenor Team: Acknowledged. Document and push curators to use a smaller maxTick as it's also used to discount the fixed tokens at max rate for collateral use. Related to that, we massively reduced the gas cost of the bestBid function and would appreciate if you could validate the fix, which is using an optimized solady's library function 1 for 1. We fuzzed all the possible values against the last and new implementation and they should be identical:

<https://github.com/Shippooor-Labs/tenor-amm/blob/6a1ac868b5b243a3fadbdfafd6fada78d4a251ec/src/types/TickBitmap.sol#L28>.

L-06 | Missing Multicall In PoolManager Contract

Category	Severity	Location	Status
Warning	● Low	PoolManager.sol	Acknowledged

Description

Tenor’s PoolManager contract currently does not implement a multicall interface, preventing users from batching multiple actions (such as adding/removing liquidity, creating/canceling limit orders and executing market orders) in a single transaction.

Without multicall, each step must be sent as a separate on-chain transaction, incurring higher gas usage and exposing users to front-running or partial state changes between steps.

Recommendation

Implement a multicall(bytes[] calldata data) function allowing users to bundle multiple PoolManager calls in a single on-chain transaction.

This approach is similar to many existing DeFi contracts, reduces gas costs and ensures atomic execution of user strategies. It would also simplify user flows and protect them against partial updates in between steps.

Resolution

Tenor Team: Acknowledged. We will have a custom bundler implementation to batch on-chain actions later on along with the market implementation.

L-07 | Unused Swap Callback

Category	Severity	Location	Status
Warning	● Low	PoolManagerHook.sol,beforeswap-parameters	Acknowledged

Description

In the PoolManagerHook implementation, the beforeSwap function includes a final bytes calldata parameter intended for user-supplied callback data.

However, the code ignores this argument altogether, making it impossible to execute the swap callback.

Because the callbackData is never referenced, any attempt to pass a payload for on-chain mid-swap actions is silently discarded.

Users cannot embed special parameters or triggers in the swap call. This deviates from the standard Uniswap V4 design that typically allows arbitrary hook data for advanced strategies.

Recommendation

Update the beforeSwap function to handle or forward the bytes calldata to the _maybeExecuteSwapCallback internal function.

Resolution

Tenor Team: Acknowledged. This parameter is only used to send arbitrary data to the custom hook implementation as per uniswap documentation. We have no use for it, so we simply don't use it. The recommendation seem to intertwine two different concepts.

L-08 | Frontrun Vault Update For Risk-Free Profit

Category	Severity	Location	Status
MEV	● Low	Global	Acknowledged

Description

In Tenor, the exchange rate for swaps is determined by a combination of the tick’s fixed interest rate and the `loanToAsset` ratio. The `loanToAsset` ratio is derived from the `ERC4626` vault that backs the loan token and typically increases over time.

However, it can drop abruptly if the vault realizes bad debt, such as during a liquidation event. This creates a clear frontrunning opportunity for attackers.

By observing an impending vault state change that will negatively affect `loanToAsset`, an attacker can execute a trade just before the update and unwind it just after—locking in a guaranteed profit at the expense of liquidity providers.

Example attack scenario:

1. A Tenor pool uses a Morpho USDC Vault as the loan token, where `loanToAsset` = 1.1
2. An attacker anticipates a pending vault liquidation that will introduce bad debt
3. The attacker front-runs with a `loanToFixed` swap, locking in `loanToAsset` = 1.1
4. Bad debt is realized → `loanToAsset` drops to 1.0.
5. The attacker back-runs with a `fixedToLoan` swap, extracting more loan tokens than initially deposited

This results in guaranteed profit for the attacker, while the liquidity providers bear the loss.

Recommendation

For limit order liquidity, one option is to consider adding per-order slippage protection. This could be done by storing the user-defined `slippageLimit` at order creation, and validate it at execution.

However, this will require redesigning the current batching system which aggregates orders from different users. For liquidity providers, implementing similar protections will be more complex and may require a separate mechanism or design discussion.

Resolution

Tenor Team: Acknowledged. We document this behavior and specify that users should not use a `ERC4626` with an exchange rate that can go down. Our market implementation handles bad debt accrual to prevent this, so does Metamorpho 1.1.

L-09 | Lack Of Incentives To Lend In Tenor

Category	Severity	Location	Status
Logical Error	● Low	BasePoolManager.sol: 195	Acknowledged

Description

During a pool initialization, the `bpsPerTick` and `maxTick` are cached in the pool state. The max APY that a user can receive when lending is `bpsPerTick * maxTick`.

However, if the `loanToken` is a very profitable ERC4626 vault, there could be cases where there are no incentives to keep lending or providing liquidity in Tenor protocol, as the max fixed APY available is way lower than the one received by just holding the vault tokens.

Recommendation

If this is intended behavior, document this to the users so they are aware of it and set the appropriate key during initialization. Alternatively, consider setting a min value for `bpsPerTick` and `maxTick` so the max APY is high enough to cover all scenarios.

Resolution

Tenor Team: Acknowledged. This is intended behavior and will be documented.

L-10 | Incorrect Fee Accounting During exactIn Swaps

Category	Severity	Location	Status
Logical Error	● Low	SwapUtils.sol: 176-191	Resolved

Description

During `exactIn` swaps, the `calculateAmountOut` function is used to determine the output amount. If `userAmountOut` is greater than `totalTickBalance`, the tick will be fully cleared during the swap. In that case, `userAmountOut` will be set to `totalTickBalance`, and `calculateAmountIn` will be used to determine `userAmountIn` since the output amount is known.

The newly calculated `userAmountIn_` must be less than or equal to the actual `userAmountIn`. However, there is an edge case where `userAmountIn_` exceeds the `userAmountIn` due to rounding. In this case, `userRemainingIn` is set to 0.

However, while setting `userRemainingIn` to 0 handles the remaining amount, `poolFeesPaidCurrencyIn` is not properly handled. In this scenario, the returned `poolFeesPaidCurrencyIn` value comes directly from the `calculateAmountIn` function, which was calculated using the higher `userAmountIn_`, rather than the actual `userAmountIn`.

As a result, `poolFeesPaidCurrencyIn` will be higher than it should be, leading to a lower `tickAmountIn` value during the swap.

Recommendation

`poolFeesPaidCurrencyIn` should be adjusted when `userAmountIn_ > userAmountIn`.

Resolution

Tenor Team: The issue was resolved in commit [603746e](#).

L-11 | Incorrect Errors Are Used In PoolAccounting

Category	Severity	Location	Status
Informational	● Low	PoolAccounting.sol: 54-55	Resolved

Description

In the `updatePoolAccounting` function, the `InsufficientPoolLoanBalance` error is used when the `fixed` token balance is insufficient, and the `InsufficientPoolFixedBalance` error is used when the `loan` token balance is insufficient.

Recommendation

Swap the errors to match their corresponding tokens.

Resolution

Tenor Team: The issue was resolved in commit [edab1c8](#).

L-12 | No Callback Support For Liquidity Operations

Category	Severity	Location	Status
Informational	● Low	BasePoolManager.sol	Acknowledged

Description

Currently, when users add or withdraw liquidity from a Tenor pool, there is no mechanism to pass `callbackData`. As a result, the `onTenorSwapCallback` hook is never triggered during these operations.

In contrast, swap operations allow users to supply `callbackData`, enabling advanced interactions like flash accounting, dynamic approvals, or token sourcing from smart contracts.

The absence of callback support for liquidity actions reduces the composability of the Tenor protocol and may limit integration opportunities with other on-chain protocols or vault systems.

Recommendation

Introduce an optional `callbackData` parameter for add/withdraw liquidity functions and trigger `onTenorSwapCallback` if data is provided.

Resolution

Tenor Team: Acknowledged. The tokenized liquidity is not transferable and we do not see a lot of value in adding this callback.

L-13 | Misleading Event Emission

Category	Severity	Location	Status
Informational	● Low	PoolManagerHook.sol: 61, 64	Acknowledged

Description

Users can perform swaps directly through Uniswap, and the Uniswap router is expected to be used for this. The actual swap will occur during the `swapInternal` call in the `beforeSwap` hook.

The `sender` value is used for both the `receiver` and `user` parameters. However, during this call, `sender` refers to the Uniswap router address, not the actual user.

Although these addresses are not directly used for token transfers, the `Swap` event will be emitted with the router address instead of the actual user or receiver.

Recommendation

Be aware of this behavior when using event listeners. Alternatively, consider passing the user address via `hookdata` when performing swaps through the router.

Resolution

Tenor Team: Acknowledged.

L-14 | Withdrawing 0 Shares Is Possible

Category	Severity	Location	Status
Informational	● Low	BasePoolManager.sol: 319-335	Resolved

Description

In general, the protocol does not allow minting or withdrawing zero shares. However, this restriction does not apply when withdrawing limit orders. The `withdrawLimitOrder` function does not include a check for a zero share amount and successfully executes even when the amount is zero.

Recommendation

Return early or revert if share amount is zero.

Resolution

Tenor Team: The issue was resolved in commit [4890806](#).

L-15 | ERC6909 Approvals And Operator Ignored

Category	Severity	Location	Status
Access Control	● Low	BasePoolManager.sol	Acknowledged

Description

The ERC6909 token standard allows an account to delegate control by approving another account or assigning an operator. However, in the current implementation, only the token owner is permitted to perform actions such as withdrawing liquidity or cancelling limit orders.

As a result, allowance and operator are effectively ignored, which deviates from expected ERC6909 behavior and limits composability.

Recommendation

Implement permission checks to allow both approved accounts and operators to act on behalf of the token owner.

Resolution

Tenor Team: Acknowledged. Saying that allowance and operator are effectively ignored, deviating from the expected ERC6909 behavior is false. EIP-6909 states that the the allowance/operator are only granting unlimited transfer permissions, which is the case here, although transferring liquidity is not allowed. Using the liquidity token allowance/operator to permit withdrawing limit orders directly within the pool manager would introduce an unclear behavior which is out of scope for this EIP. Withdrawing limit orders on behalf of user is already possible as the liquidity tokens can be transferred from and withdrawn by the operator.

L-16 | loanToAsset Does Not Account For Slippage/Fees

Category	Severity	Location	Status
Warning	● Low	ERC4626Utils.sol	Acknowledged

Description

In the custom ERC4626Utils library, the function loanToAsset derives the exchange rate via:

```
exchangeRate = loanToken.convertToAssets(10 ** (EXCHANGE_RATE_DECIMALS +
loanTokenDecimals - underlyingTokenDecimals));
```

However, per the [ERC4626 specification](#), convertToAssets does not account for withdrawal fees or slippage. As a result, it may overestimate the actual amount of assets received upon redemption.

In contrast, the previewRedeem function is designed to return the actual amount of assets, inclusive of slippage and fees.

While this has no immediate effect when using Morpho vaults (which do not charge fees), it could cause mispricing or unexpected behavior if the protocol integrates with fee-charging or slippage-prone vaults in the future.

Recommendation

Use previewRedeem instead of convertToAssets to get an accurate, fee-adjusted conversion rate.

Resolution

Tenor Team: Acknowledged. This introduces weird scenarios if the fees are subject to change, but will be documented.

L-17 | Unused Error In PoolActions

Category	Severity	Location	Status
Informational	● Low	PoolActions.sol: 7	Resolved

Description

InvalidSwap error in PoolActions.sol library is not used and can be removed.

Recommendation

Consider removing unused import.

Resolution

Tenor Team: The issue was resolved in commit [bed02a5](#).

L-18 | Missing Max bpsPerTick Check In Init Flow

Category	Severity	Location	Status
Validation	● Low	Global	Acknowledged

Description

If the PoolManagerHook contract is used the protocol checks that the given bpsPerTick value is not too big during the initialize flow in the assertCompatibilityWithUniswap function.

As the bpsPerTick value also equals the swap fee it makes a lot of sense to not allow unreasonable values here. This check is missing in the pure Tenor PoolManager.initialize flow.

Recommendation

Consider adding the maximum check for the given bpsPerTick value to the initializeInternal function and removing it from the assertCompatibilityWithUniswap function.

Resolution

Tenor Team: Acknowledged. This is the expected behavior as we don't want to restrict uses cases. In the hook version, we have a hard requirement as the pool key doesn't allow more. This will be documented.

L-19 | Functions Naming Convention

Category	Severity	Location	Status
Best Practices	● Low	Global	Resolved

Description

The protocol uses names starting with an underscore for private functions, but not for internal functions. This naming style goes against the solidity convention for external/public vs internal/private functions, more details [here](#)

Recommendation

Consider adapting to the function naming convention, according to the solidity docs shared above.

Resolution

Tenor Team: The issue was resolved in commit [7275dcb](#).

L-20 | Mismatched Comments For Fee And Tick Spacing

Category	Severity	Location	Status
Documentation	● Low	HookPoolKey.sol: 87	Resolved

Description

The HookPoolKey.toUniswapKey converts a Tenor pool key to a Uniswap pool key. There are some comments explaining the uniswap fee and tickSpacing encoding.

However, the first comment mentions Uniswap pool fee, even though the encoding is for tickSpacing. Same happens with the fee param below.

Recommendation

Update the comments to clearly show which parameter is being explained (first one is about tick spacing, second about fee). This avoids confusing the reader.

Resolution

Tenor Team: The issue was resolved in commit [84e8bdc](#).

L-21 | External Incentives Are Lost

Category	Severity	Location	Status
Rewards	● Low	Global	Acknowledged

Description

According to Morpho docs, Users automatically earn rewards while participating in incentivized markets or vaults. Therefore, the PoolManager will be entitled to claim these rewards as it holds vault tokens.

Although anyone can claim on behalf of other addresses, the current implementation does not contain a function to extract these rewards, so they are effectively lost.

For the case of PoolManagerHook the Morpho vault tokens will be stored in the Uniswap side, so these can't never be retrieved.

Recommendation

Consider implementing an owner function to withdraw these extra incentives.

Resolution

Tenor Team: Acknowledged. This is a known behavior and will be documented. Tenor's implementation has a loan token that can claim rewards.

L-22 | Typos In Protocol Documentation

Category	Severity	Location	Status
Informational	● Low	Protocol documentation	Resolved

Description

There are multiple typos / small mistakes in the protocol documentation.

Recommendation

Consider fixing all the typos suggested.

Resolution

Tenor Team: Resolved. Fixed in the documentation website repository.

L-23 | Uniswap V4 DoS During Callback

Category	Severity	Location	Status
DoS	● Low	PoolManagerHook.sol: 138-159	Acknowledged

Description

The flow of the protocols `_unlockCallback` function looks like the following:

- settle positive delta (funds are pushed to user)
- optional callback is executed
- settle negative delta (funds are pulled from the user)

As this optional callback happens during the `_unlockCallback` flow the Uniswap V4 `PoolManager` contract is already unlocked.

This means that it is not possible for the user to use the Uniswap Router to interact with the Uniswap V4 system (for example to perform a swap) as this interaction would revert with a `AlreadyUnlocked` error.

It is possible for the user to interact directly with the Uniswap V4 core contracts, however this requires the user to be experienced enough with Uniswap V4 to do so.

Recommendation

Consider performing the callback while Uniswap V4 is locked, or document this behaviour.

Resolution

Tenor Team: Acknowledged. The goal of this callback is to be able mint the exact amount of fixed tokens needed in a swap. People wanting to use Uniswap will go through the Uniswap routing directly.

L-24 | Rethink Uniswap Integration

Category	Severity	Location	Status
Logical Error	● Low	PoolManagerHook.sol	Acknowledged

Description

The protocol allows to deploy two different types of pools:

- The pure tenor pool named PoolManager
- A similar version that also interacts with Uniswap named PoolManagerHook

The protocol stated out that there are two reasons for the Uniswap integration:

1: To allow swaps over Uniswap

As Uniswap is only used for accounting and no real swaps take place a user who performs a swap over Uniswap will likely have a very bad user experience. The shown price at Uniswap will never change after init therefore the user sees one price in the UI and swaps at a totally different price. This could be very confusing and lead to setting a dangerous slippage check.

2: "If Uniswap creates an "earn" product they could simply route the volume using the hooked version of the pools"

As now swaps take place on Uniswap the volume routed through it will always be 0 and Uniswap is not able to charge any fees. This makes it unlikely that Uniswap will distribute rewards to these pools.

Recommendation

Rethink if the Uniswap version makes sense or if it just adds more potentially vulnerable code without a real benefit.

Resolution

Tenor Team: Acknowledged. We're keeping both implementations to allow more uses cases in the future.

L-25 | Lack Of Incentives To Provide Liquidity

Category	Severity	Location	Status
Validation	● Low	PoolManagerRoles.sol: 149	Acknowledged

Description

The pool owner can use `setFeeShares` to adjust the share of the fees earned by the liquidity providers and limit order creators (market makers).

The current implementation only validate if these are less than or equal to 100. However, allowing 100% fee share does not make much sense, as market makers will be earning zero fees. Therefore, there are no incentives to provide liquidity to the pool.

Recommendation

Consider setting the max fee share to a value below 100%.

Resolution

Tenor Team: Acknowledged. Don't want to restrict use cases, although setting this to 100 will push LPs away naturally.

L-26 | Failed Invariants For Edge Cases

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Resolved

Description

The following invariants were invalidated:

- SWP_01: will always fail when dealing with limit orders, either by swapping and creating one, or filling a limit order during swap. This was more like an issue with the invariant implementation, as unfilled, partial and fulfilled shared will change before and after the swap.
- SWP_02: fails when doing a swap-limit order, when spot tick only contains limit orders (as fulfilled limit orders are removed from bitmap)
- SWP_06: when swapping loan to fixed at tick 1, cases where you receive less fixed tokens than the amount of loan tokens swapped, therefore $\ln(\text{fixed}/\text{loan})$ is negative.
- SWP_07: swapping with very low amounts (wei), can result in implied rates above max.
- SWP_08: low underlying decimals (i.e. decimals = 1), and swapping dust amount of fixed to loan, leads to an exchange rate below 1e18.
- G_03: the pool had loan token decimals 1 and fixed token decimals 18.
- G_05: simulation reverts due to $\text{loanToAsset} = 0$ (very low asset amount in vault so $\text{convertToAssets}(1e18)$ yields to 0)

Recommendation

Document this edge cases so they can be avoided while creating a market/pool.

Resolution

Tenor Team: Ack, will document accordingly.

L-27 | Liquidity Provision Is Capital Inefficient

Category	Severity	Location	Status
Rewards	● Low	Global	Acknowledged

Description

Usually, LPs can provide X liquidity to a protocol (e.g. Uniswap, Aave, ...) and earn Y% yield from the X liquidity. For example:

- LP provides \$100k liquidity
- After one year the LP earned \$3000 (3%)

But at Tenor the LPs need to first mint a specific fixed token for the given pool by depositing collateral. Therefore they do not earn Y% yield on their given amount of X liquidity as the debt position must be overcollateralized. In case of an 80% LLTV (given as example in the docs):

- LP wants to provide \$100k liquidity
- \$50k is provided in loan tokens and yields \$1500 after one year
- The other \$50k are used as collateral to borrow \$40k fixed tokens which yield \$1200 after one year
- Therefore after one year the LP earned \$1500 + \$1200 = \$2700 (only 2.7%)

This is capital inefficient and can lead to LPs searching for better opportunities, which could result in a lack of liquidity for the protocol.

Recommendation

Consider rethinking the LP mechanics if this leads to too a lack of liquidity on Tenor.

Resolution

Tenor Team: Acknowledged. Not only the LPs can add liquidity as loan, earning interest, but they can also add fixed, which is effectively a lending position earning interest (can swap loan for fixed directly). The borrowers will mint the fixed token along with the debt, and trade the fixed token for loan to effectively borrow.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit <https://guardianaudits.com>

To view our audit portfolio, visit <https://github.com/guardianaudits>

To book an audit, message <https://t.me/guardianaudits>