



**SMART CONTRACT SECURITY AUDIT OF**



**POOLSHARK**

# Summary

**Audit Firm:** Guardian Audits

**Client Firm:** Poolshark

**Prepared By:** Owen Thurm, Daniel Gelfand, 0xKato, Kiki, devScrooge, 0xScourgeDev

**Final Report Date - August 21, 2023**

## Audit Summary

Poolshark engaged Guardian to review the security of their Directional AMM Limit Pool. From the 18th of July to the 18th of August, a team of 6 security researchers reviewed the source code in scope. The auditing approach championed manual analysis to uncover novel exploits and verify intended behavior with supporting verification from fuzzing with [Echidna](#). All invariants, findings, and remediations have been recorded in the following report.

**Issues Detected** Throughout the course of the audit numerous high impact issues were uncovered and promptly remediated by the Poolshark team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the Directional AMM Limit Pool.

**Code Quality** From the 18th of July to the 18th of August, the codebase quality improved considerably. However, it is recommended to improve in-code documentation supporting [NatSpec](#) standards and to update the [Poolshark Whitepaper](#). Additionally, given the scope of changes made to the codebase and number of critical issues detected, Guardian supports an independent security review of the protocol at a finalized frozen commit.

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

# Table of Contents

## Project Information

Project Overview ..... 4

Audit Scope & Methodology ..... 5

LimitPool UML ..... 9

LimitPoolFactory UML ..... 10

Invariants Assessed ..... 11

## Smart Contract Risk Assessment

Findings & Resolutions ..... 14

## Addendum

Disclaimer ..... 95

About Guardian Audits ..... 96

# Project Overview





## Project Summary

Project Name	Poolshark
Language	Solidity
Codebase	<a href="https://github.com/poolshark-protocol/limit">https://github.com/poolshark-protocol/limit</a>
Initial Commit	<a href="https://github.com/poolshark-protocol/limit/commit/6ab8d15c6ef9981291224c615b73f118851e6415">6ab8d15c6ef9981291224c615b73f118851e6415</a>
Final Commit	TBD

## Audit Summary

Delivery Date	August 21, 2023
Audit Methodology	Manual Review, Static Analysis, Contract Fuzzing, Symbolic Execution

## Vulnerability Summary

Vulnerability Level	Total	Acknowledged	Declined	Acknowledged	Partially Resolved	Resolved
 Critical	15	0	0	0	0	15
 High	9	0	0	0	0	9
 Medium	13	0	0	3	0	10
 Low	38	0	0	27	0	11

# Audit Scope & Methodology

## Scope

ID	File	Final SHA-1 Checksum(s)
LMP	LimitPool.sol	0537e2602b675ef1349515dd3d75d48ab632cb52
LPF	LimitPoolFactory.sol	98a2ac888a6231136b4b6132ef9fbf2cdc966877
LMPE	LimitPoolEvents.sol	b99461ed5841ecf53faa245a07df8885da703585
LPFE	LimitPoolFactoryEvents.sol	62cd37dcf871108cba8bc285b1fcc656bd458504
LPME	LimitPoolManagerEvents.sol	535b0ace07ec94feb3067e5c68b639164ed9369f
LPFST	LimitPoolFactoryStorage.sol	dcddb74749eb4d67b77e1e2d4b682a28b4f6d0
LPI	LimitPoolImmutables.sol	31fecbd60921578acec28976ddc720d327fbf6ce
LPS	LimitPoolStorage.sol	94ae085130ec3981d04c69f470b00aa76810ea93
RPEI	RangePoolERC1155Immutables.sol	44d7ae5679bd574aeef252502356111179ccae4e
LPFS	LimitPoolFactoryStructs.sol	0172e649882f7c6cbaacef3d506bba5746adec45
PS	PoolsharkStructs.sol	86f70fdafce4af666005e8d60016813d432b43da
REG	ReentrancyGuard.sol	8a2bc72b0a01ee2dca4377e6784f39b76278b4f2
TMAP	TickMap.sol	6a75bd6e2aca4aa189bb45c9f932fd424ae66b43
TK	Ticks.sol	6a75bd6e2aca4aa189bb45c9f932fd424ae66b43
CLAIMS	Claims.sol	c786b377719ddabba9e813cc8f375f5c1e3362cb
EMAP	EpochMap.sol	17f414d5f30eb116ad3ed3c2686dea6fae4e58bc
POS	LimitPositions.sol	a997686c05b2e1f69ff81d0e3807dcf0af5e230a

# Audit Scope & Methodology

## Scope

ID	File	Final SHA-1 Checksum(s)
LT	LimitTicks.sol	94568aa3feb5f3f26767e7345f4ea1d370fe52ba
BCALL	BurnLimitCall.sol	6f30aae0fd8febe7371c1ae084a6429a5912ac3d
MCALL	MintLimitCall.sol	615657ec5c2884995276ef172f2cffd7fc2adea9
CP	ConstantProduct.sol	2cf31995d6c332993d2029fcb0ce948c6f05a12b
OM	OverflowMath.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
QC	QuoteCall.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
SC	SampleCall.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
SWPC	SwapCall.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
RNP	RangePositions.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
RTK	RangeTicks.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
RNK	RangeTokens.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
SAMP	Samples.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
FMATH	FeeMath.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
BRC	BurnRangeCall.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
MRC	MintRangeCall.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
COL	Collect.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
SAFEC	SafeCast.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
SAFET	SafeTransfers.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641

# Audit Scope & Methodology

## Scope

ID	File	Final SHA-1 Checksum(s)
SFST	SafeState.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
STR	String.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
LPE	LimitPoolErrors.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
LMPM	LimitPoolManager.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
PROU	PoolRouter.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
RP1155	RangePoolERC1155.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
RPE	RangePoolErrors.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641
RLIB	RebaseLibrary.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641

# Audit Scope & Methodology

## Methodology

The auditing process pays special attention to the following considerations:

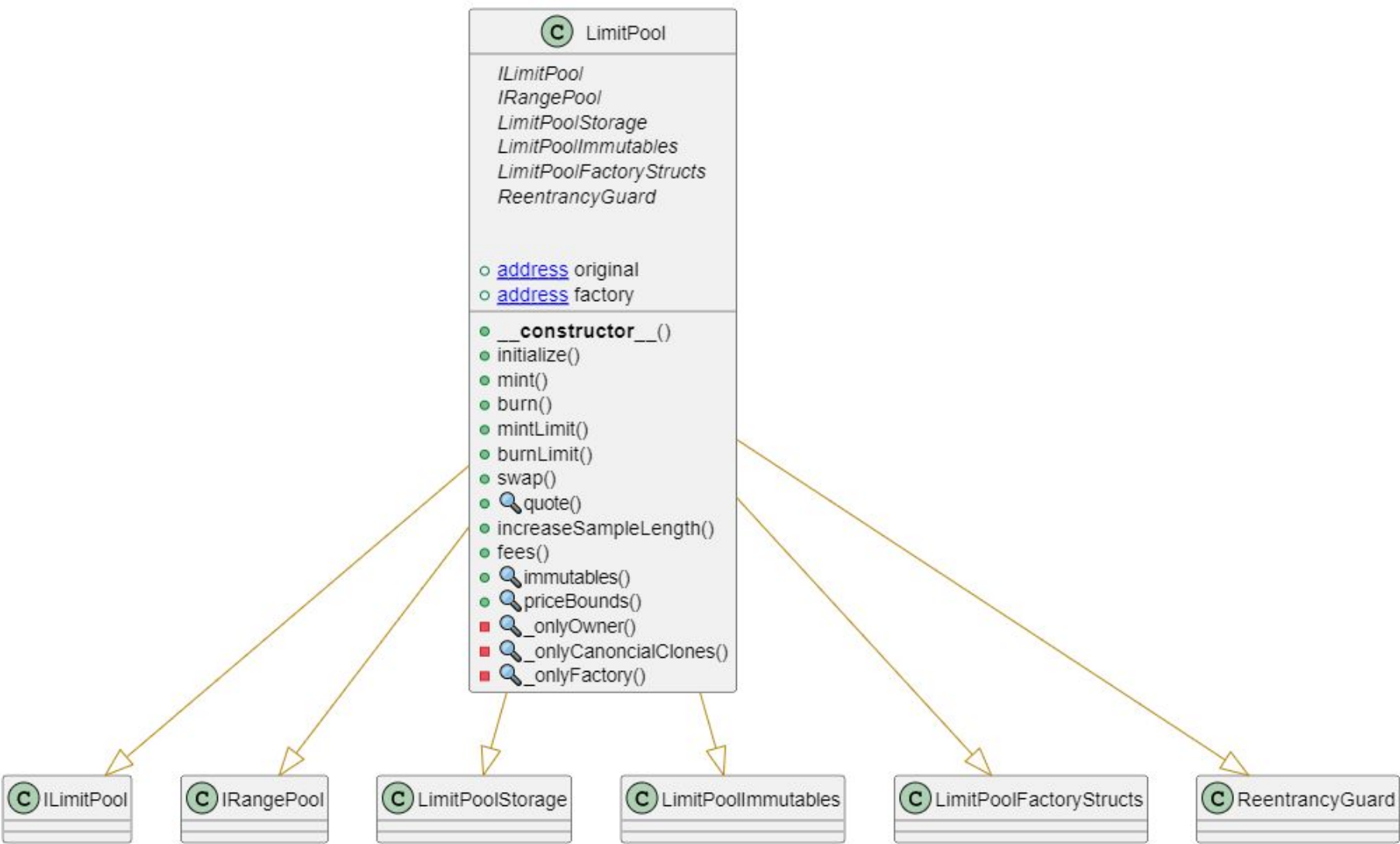
- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
- Contract fuzzing for verification of intended invariants.
- Symbolic Execution for verification of intended behavior.

## Vulnerability Classifications

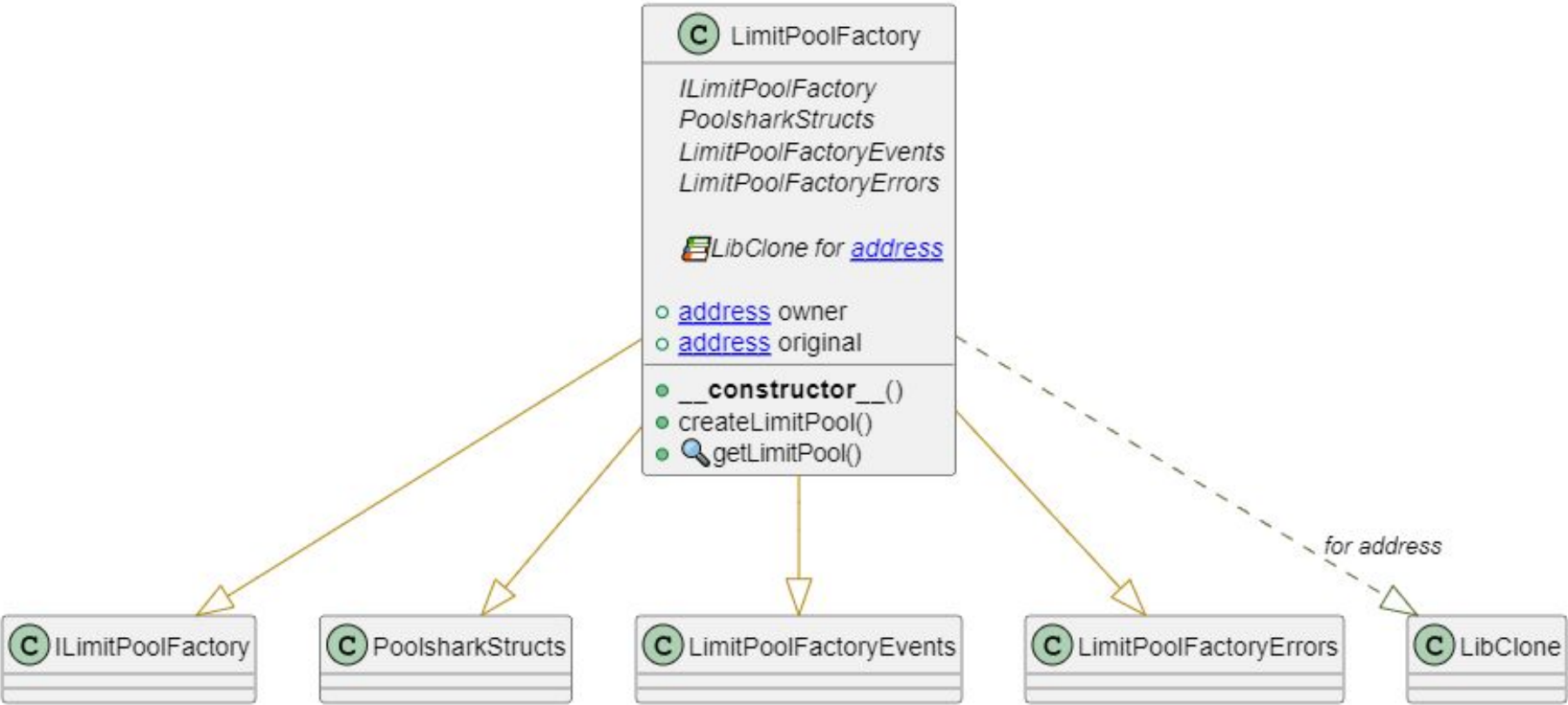
Vulnerability Level	Classification
● Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
● High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
● Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
● Low	Minor deviation from best practices.



# UML Diagram - LimitPool



# UML Diagram - LimitPoolFactory



# Invariants Assessed

During Guardian's review of Poolshark's Directional AMM Limit Pool, fuzz-testing with [Echidna](#) was performed on the protocol's main functions. Given the dynamic interactions and the potential for unforeseen edge cases in the Limit Pool, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 300,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Definition	Run Count
<u>GLOBAL-1</u>	The upper position boundary is above the lower boundary	<code>lower &lt; upper</code>	100,000,000+
<u>GLOBAL-2</u>	The lower and upper ticks are on full ticks	<code>lower % tickSpacing == 0</code> <code>upper % tickSpacing == 0</code>	100,000,000+
<u>GLOBAL-3</u>	The price of pool0 is always greater than or equal to the price of pool1	<code>price0 &gt;= price1</code>	100,000,000+
<u>GLOBAL-4</u>	Pool liquidity never underflows	<code>pool liquidity &gt;= decrement amount</code>	100,000,000+
<u>GLOBAL-5</u>	liquidityGlobal never underflows	<code>liquidityGlobal &gt;= decrement amount</code>	100,000,000+
<u>GLOBAL-6</u>	liquidityAbsolute never underflows	<code>liquidityAbsolute &gt;= decrement amount</code>	50,000,000+
<u>GLOBAL-7</u>	Pool liquidity never overflows	<code>liquidity + amount &lt;= type(uint128).max</code>	100,000,000+
<u>GLOBAL-8</u>	Transfer amount never exceeds pool balance	<code>poolBalance &gt;= outputAmount</code>	100,000,000+
<u>GLOBAL-9</u>	Full ticks have a priceAt of 0	<code>If (tick % tickSpacing == 0)</code> <code>priceAt == 0</code>	100,000,000+
<u>MINT-1</u>	Global liquidity increases when minting a position	<code>liquidityGlobalBefore &gt; liquidityGlobalAfter</code>	50,000,000+

# Invariants Assessed

ID	Description	Definition	Run Count
<u>MINT-2</u>	Pool liquidity is non-zero on undercut	If (undercut) liquidityAfter > 0	50,000,000+
<u>MINT-3</u>	Minting a position then burning the position causes no change in global liquidity	mint() burn() liquidityGlobalAfter == liquidityGlobalBefore	25,000,000+
<u>MINT-4</u>	liquidityDelta is always less than or equal to liquidityAbsolute	liquidityDeltaOnTick <= liquidityAbsoluteOnTick	25,000,000+
<u>MINT-5</u>	liquidityAbsolute increases when not undercutting price	If (undercut) liquidityAbsoluteAfter > liquidityAbsoluteBefore	25,000,000+
<u>MINT-6</u>	Pool liquidity is positive on unlock	unlock() pool liquidity > 0	50,000,000+
<u>BURN-1</u>	Burning a position decreases global liquidity	liquidityGlobalBefore < liquidityGlobalAfter	50,000,000+
<u>TMAP-1</u>	Tick exists if set twice	set() set() get() == true	1,000,000+
<u>TMAP-2</u>	Tick does not exist if set the unset	set() unset() get() == false	1,000,000+
<u>TMAP-3</u>	Tick exists when set	set() get() == true	1,000,000+
<u>TMAP-4</u>	Next tick includes half tick when inclusive	If (tick % tickSpacing >= halfTickSpacing) If (inclusive) halfTick == next(tick)	1,000,000+

# Invariants Assessed

ID	Description	Definition	Run Count
<u>TMAP-5</u>	Tick does not exist when unset	unset() get() == false	1,000,000+
<u>TMAP-6</u>	Previous tick includes half/full tick when inclusive	If (tick % halfTickSpacing == 0) If (inclusive) tick == previous(tick)	1,000,000+

# Findings & Resolutions

ID	Title	Category	Severity	Status
<u>GLOBAL-1</u>	Shared TickMap Errantly Unsets Ticks	Logical Error	● Critical	Resolved
<u>CLAIMS-1</u>	Unset End Tick Allows Malicious Claims	Logical Error	● Critical	Resolved
<u>MCALL-1</u>	Pool State Unsaved Leading To Underflow	Underflow	● Critical	Resolved
<u>BCALL-1</u>	Overwritten Position On Remove	Logical Error	● Critical	Resolved
<u>CLAIMS-2</u>	Pool Bricked Due To Null Position	Logical Error	● Critical	Resolved
<u>TK-1</u>	Pool Liquidity Double Counted	Logical Error	● Critical	Resolved
<u>PROU-1</u>	Stolen Approvals	Access Control	● Critical	Resolved
<u>TMAP-1</u>	Broken Swap Due To Incorrect Cross Tick	Logical Error	● Critical	Resolved
<u>TK-2</u>	Liquidity Underflow Due To Tick Rounding	Rounding	● Critical	Resolved
<u>TK-3</u>	Cross Tick Skips Half Ticks	Logical Error	● Critical	Resolved
<u>LMP-1</u>	Lack Of Access Restriction For Initialize Function	Access Control	● Critical	Resolved
<u>TK-4</u>	Users Can Maliciously Claim At The Current Pool Price	Logical Error	● Critical	Resolved
<u>TK-5</u>	Half Tick Liquidity Never Unlocked	Logical Error	● Critical	Resolved

# Findings & Resolutions

ID	Title	Category	Severity		Status
<u>TK-6</u>	Pool.price Not Updated In Ticks.unlock	Logical Error	●	Critical	Resolved
<u>TK-7</u>	Swaps Bricked Due To Malicious Position	Logical Error	●	High	Resolved
<u>CLAIMS-3</u>	Position Resized To Half Tick	Logical Error	●	High	Resolved
<u>LMP-2</u>	Unclaimable Fees	Logical Error	●	High	Resolved
<u>TMAP-2</u>	Incorrect Tick Rounding	Rounding	●	High	Resolved
<u>GLOBAL-2</u>	Odd Tick Spacing Should Not Be Used	Configuration	●	High	Resolved
<u>POS-1</u>	Yield Can Be Stolen From Liquidity Providers	Logical Error	●	High	Resolved
<u>POS-2</u>	liquidityGlobal Not Decrementd	Logical Error	●	High	Resolved
<u>CLAIMS-4</u>	Position Overwritten At Claim Tick	Logical Error	●	High	Resolved
<u>CLAIMS-5</u>	Users Prevented From Burning	Logical Error	●	High	Resolved
<u>TK-8</u>	exactOut Does Not Function As Expected	Logical Error	●	High	Resolved
<u>LMP-3</u>	protocolFee0 Overwritten	Logical Error	●	Medium	Resolved
<u>GLOBAL-3</u>	Fee-On-Transfer Tokens	Fee-on-transfer	●	Medium	Acknowledged

# Findings & Resolutions

ID	Title	Category	Severity	Status
<u>POS-3</u>	Small Prices Round Out Of Range When Multiplied	Rounding	● Medium	Acknowledged
<u>POS-4</u>	Resizes Without Swaps Are Too Severe	Logical Error	● Medium	Resolved
<u>POS-5</u>	Misleading liquidityBurned emitted	Events	● Medium	Resolved
<u>POS-6</u>	Position Minted With 0 Liquidity	Logical Error	● Medium	Resolved
<u>FMATH-1</u>	Inaccurate Fee On Event	Events	● Medium	Resolved
<u>TMAP-3</u>	Inclusive Skips Half Tick On Next	Logical Error	● Medium	Resolved
<u>BCALL-2</u>	State Saved After Token Transfer	Reentrancy	● Medium	Resolved
<u>LMP-4</u>	Quote and Snapshot Read-only Reentrancy Risk	Reentrancy	● Medium	Resolved
<u>RPEI-1</u>	Missing ERC1155 Support Validation	Validation	● Medium	Resolved
<u>PROU-2</u>	Lacking Slippage Controls	Slippage	● Medium	Acknowledged
<u>LPF-1</u>	Lack Of Token Validation	Validation	● Low	Resolved
<u>POS-7</u>	Superfluous finalTick Assignment	Superfluous Code	● Low	Resolved
<u>POS-8</u>	Superfluous Else Case	Superfluous Code	● Low	Resolved



# Findings & Resolutions

ID	Title	Category	Severity	Status
<u>GLOBAL-4</u>	Unused Q96 Constant	Superfluous Code	<div><div></div>Low</div>	Resolved
<u>TK-9</u>	Typo	Typo	<div><div></div>Low</div>	Resolved
<u>TK-10</u>	Unnecessary Ternary Operator	Superfluous Code	<div><div></div>Low</div>	Resolved
<u>LPF-2</u>	Inefficient Token Assignment	Optimization	<div><div></div>Low</div>	Resolved
<u>GLOBAL-5</u>	Superfluous GlobalState Storage Variable	Superfluous Code	<div><div></div>Low</div>	Resolved
<u>POS-9</u>	Inconsistent mintPercent Decimals	Consistency	<div><div></div>Low</div>	Acknowledged
<u>LMP-5</u>	Lacking Zero Address Checks	Validation	<div><div></div>Low</div>	Acknowledged
<u>LMP-6</u>	Typo	Typo	<div><div></div>Low</div>	Acknowledged
<u>EMAP-1</u>	Redundant _tick Function	Superfluous Code	<div><div></div>Low</div>	Acknowledged
<u>RLIB-1</u>	Superfluous RebaseLibrary	Superfluous Code	<div><div></div>Low</div>	Acknowledged
<u>CLAIMS-6</u>	Superfluous claimTick Assignment	Superfluous Code	<div><div></div>Low</div>	Acknowledged
<u>TK-11</u>	Outdated Comments	Documentation	<div><div></div>Low</div>	Acknowledged
<u>CLAIMS-7</u>	Superfluous ClaimTickEpoch Assignments	Superfluous Code	<div><div></div>Low</div>	Acknowledged

# Findings & Resolutions

ID	Title	Category	Severity	Status
<u>SFST-1</u>	Unused Checks Library	Superfluous Code	● Low	Acknowledged
<u>POS-10</u>	Unreachable Code	Superfluous Code	● Low	Acknowledged
<u>CLAIMS-8</u>	Superfluous Early Return Logic	Superfluous Code	● Low	Acknowledged
<u>POS-11</u>	Superfluous Position Write	Superfluous Code	● Low	Acknowledged
<u>LMPM-1</u>	Enabled Pool Configuration Cannot Be Disabled	Configuration	● Low	Acknowledged
<u>POS-12</u>	Redundant Validation	Superfluous Code	● Low	Acknowledged
<u>GLOBAL-6</u>	Superfluous refundTo Variable	Superfluous Code	● Low	Acknowledged
<u>GLOBAL-7</u>	Floating Pragma Version	Floating Pragma	● Low	Resolved
<u>GLOBAL-8</u>	For Loop Optimizations	Optimization	● Low	Resolved
<u>POS-13</u>	Superfluous Removal Logic	Superfluous Code	● Low	Acknowledged
<u>GLOBAL-9</u>	Unsafe casting	Casting	● Low	Acknowledged
<u>GLOBAL- 10</u>	Dust Positions Negatively Impact The Protocol	Manipulation	● Low	Acknowledged
<u>CLAIMS-9</u>	Malicious Burn DoS	DoS	● Low	Acknowledged

# Findings & Resolutions

ID	Title	Category	Severity	Status
<u>RNP-1</u>	Liquidity Overflow Check Is Inconsistent	Validation	● Low	Acknowledged
<u>TK-12</u>	Superfluous cache.liquidity Adjustments	Superfluous Code	● Low	Acknowledged
<u>POS-14</u>	Position Unnecessarily Deleted	Optimization	● Low	Acknowledged
<u>CLAIMS-10</u>	Futile Burns Are Allowed	Validation	● Low	Acknowledged
<u>TK-13</u>	Unnecessary uint256 Casting	Superfluous Code	● Low	Acknowledged
<u>GLOBAL-11</u>	Redundant Boolean Logic In _empty	Superfluous Code	● Low	Acknowledged
<u>MRC-1</u>	Inefficient Validation	Validation	● Low	Acknowledged
<u>RTK-1</u>	Superfluous Else Case	Superfluous Code	● Low	Resolved
<u>GLOBAL-12</u>	Clones Can Receive ETH But Not Withdraw It	Trapped Ether	● Low	Resolved
<u>BRC-1</u>	Superfluous Addition And Subtraction	Optimization	● Low	Acknowledged

# GLOBAL-1 | Shared TickMap Errantly Unsets Ticks

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">00db8d3494769b7363007322d3b75a9975982519</a>	Global	Resolved

## Description [PoC](#)

When burning a zeroForOne position, ticks in the TickMap are unset deAcknowledged on if there is a liquidityDelta of 0 at that tick in the ticks0 mapping. However, there may be a nonzero liquidityDelta on that tick in the ticks1 mapping.

This leads to ticks being unset from one zeroForOne side when they are critical for the other. Therefore these ticks will not be crossed during a swap which invalidates the protocol accounting.

## Recommendation

Adopt two TickMaps and EpochMaps to solve this particular failure case as well as to avoid any additional potential logical errors related to sharing a TickMap and EpochMap.

## Resolution

Poolshark Team: The issue was resolved in commit [0e40b90](#).

# CLAIMS-1 | Unset End Tick Allows Malicious Claims

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">00db8d3494769b7363007322d3b75a9975982519</a>	Claims.sol: 95	Resolved

## Description [PoC](#)

During a swap a position’s end tick may become unset, leaving a position with no upper tick that is set in the `TickMap`. The pool price can then be undercut such that price is below the position’s end tick.

The user is then able to claim at the current pool price and pass the claims validation as the `TickMap.next` yields the max tick which carries an unset `epochLast`.

Ultimately this allows the user to withdraw `token0` and `token1` balances from the contract that do not correspond to their actual fill amount, invalidating the system's accounting.

## Recommendation

Always validate the position’s end tick `epochLast` in the claims validation logic with the following:

```
uint32 endTickAccumEpoch = EpochMap.get(params.zeroForOne ? params.upper : params.lower,
tickMap, constants);

if (endTickAccumEpoch > cache.position.epochLast) {
    require (false, 'WrongTickClaimedAt5()');
}
```

## Resolution

Poolshark Team: The issue was resolved in commit [0e40b90](#).

# MCALL-1 | Pool State Unsaved Leading To Underflow

Category	Severity	Commit	Location	Status
Underflow	● Critical	<a href="#">5807eb8a40561f14adf2734f27bf694519e435ab</a>	MintCall.sol: 114	Resolved

## Description [PoC](#)

The pool state is only saved if a position is minted, e.g. if `params.amount > 0 && params.lower < params.upper` is satisfied. However, liquidity for the pool can be unlocked without a position actually being minted when the `params.amount` is swapped entirely and the below case is entered.

```
if (cache.pool.liquidity == 0) {  
    /// @dev - this makes sure to have liquidity unlocked if undercutting  
  
    (cache, cache.pool) = Ticks.unlock(cache, cache.pool, ticks, tickMap, params.zeroForOne);  
}
```

As a result, liquidity can be unlocked in the pool yet never saved. Once the liquidity is unlocked, the tick where the liquidity was stashed is cleared. This leads to catastrophic consequences as the `liquidityDelta` on the cleared tick will not be crossed and added to the pool.

Ultimately, there will be liquidity underflow when the corresponding negative `liquidityDelta` is added to the `pool.liquidity` during `Ticks.unlock` or `Ticks._cross`. This breaks a key invariant that `pool.liquidity` should never underflow, disrupting all `LimitPool` operations.

## Recommendation

Save the pool state even when a position is not minted in `MintCall.perform`.

## Resolution

Poolshark Team: The recommendation was implemented in commit [00db8d3](#).

# BCALL-1 | Overwritten Position On Remove

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">f26fff27d33956e233e5d0e71a31f91c8e75f293</a>	BurnCall.sol: 60-85	Resolved

## Description [PoC](#)

When the `Positions.remove` function is called, the updated `cache.position` is not returned. Therefore at the end of the `perform` function all updates are overwritten.

```
if ((params.zeroForOne ? params.claim != params.upper
                        : params.claim != params.lower))
    params.zeroForOne
        ? positions[msg.sender][params.claim][params.upper] = cache.position
        : positions[msg.sender][params.lower][params.claim] = cache.position;
```

Consequently, this duplicates the amounts of a position, leaving liquidity in the position when there should not be any since the state of the position prior to the burn is recorded.

## Recommendation

Return the updated `cache.position` from `Positions.remove` and save this updated position state.

## Resolution

Poolshark Team: The recommendation was implemented in commit [5a30bbf](#).

# CLAIMS-2 | Pool Bricked Due To Null Position

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">91b0f3dc0e684f69825f3f8f08a3b15e875c1829</a>	Claims.sol: 335, 369	Resolved

## Description [PoC](#)

When a position has been fully filled and a user is claiming with a `params.amount` of 0 the position is resized to a range spanning 0 ticks, e.g. [100, 100], however the position remains at those ticks with a nonzero liquidity.

The user is now able to burn this remaining null position with a nonzero `params.amount`, however now the `params.lower == params.upper == params.claim`.

Therefore for `zeroForOne` positions, when `cache.pool.price < cache.priceLower`, the `cache.removeLower` will be assigned to `true`.

And for `!zeroForOne` positions, when `cache.pool.price > cache.priceLower`, `cache.removeUpper` will be assigned to `true`.

This results in the position's liquidity being subtracted from the start tick, when the liquidity had already previously been removed when the position was originally crossed into and filled.

Therefore the `startTick.liquidityDelta` will be more negative than the `pool.liquidity` at that price and result in the entire pool being bricked upon reaching the `startTick`.

## Recommendation

When users are claiming a fully filled position where the `params.claim == endTick`, always zero out the position liquidity among other attributes as the position no longer exists.

## Resolution

Poolshark Team: The recommendation was implemented in commit [3682ae9](#).



# TK-1 | Pool Liquidity Double Counted

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">e592dd31338bef73878ccc95ceb9c9821d0db9e8</a>	Ticks.sol: 490	Resolved

## Description [PoC](#)

In the Ticks.insertSingle function, the pool.liquidity is often zeroed out and stashed onto the tickToSave. However in some cases the pool.liquidity will not be zeroed out, and yet it is still stashed on the tickToSave.

This means that the pool.liquidity is immediately double counted in both the active liquidity and the tick.liquidityDelta. Now if this tick is crossed this liquidityDelta will be added to the pool.liquidity and the pool will have double the liquidity than it ought to at the current price.

Therefore users will be able to swap using more liquidity than exists at the current range, and they will draw from tokens in positions that are far away from the current pool price. This is catastrophic, as users will not be able to burn their positions as their underlying token amounts have been improperly swapped at the current price, leaving a lack of tokenIn to withdraw.

## Recommendation

Ensure that whenever the liquidityDelta is updated on the tickToSave, the pool.liquidity is zeroed out, every time. Include the zeroing out of the pool.liquidity exactly where the tick.liquidityDelta is incremented on line 490.

## Resolution

Poolshark Team: The recommendation was implemented in commit [d5443ff](#).

# PROU-1 | Stolen Approvals

Category	Severity	Commit	Location	Status
Access Control	● Critical	<a href="#">6ab8d15c6ef9981291224c615b73f118851e6415</a>	PoolRouter.sol: 24	Resolved

## Description

The poolsharkSwapCallback function on the PoolRouter contract does not validate that the msg.sender is a real Poolshark limit pool and yet gives the msg.sender the ability to transfer from any address to the msg.sender.

Therefore any user that has approved the PoolRouter contract may have their approval amounts stolen by an arbitrary address invoking the poolsharkSwapCallback.

## Recommendation

Validate that the msg.sender is indeed a registered Poolshark LimitPool upon invocation of the poolsharkSwapCallback function.

## Resolution

Poolshark Team: The recommendation was implemented in commit [593ff6a](#).

# TMAP-1 | Broken Swap Due To Incorrect Cross Tick

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">d5443ffcf700279ce449da07e44624c2d04286e7</a>	TickMap.sol: 67-68	Resolved

## Description [PoC](#)

Because the TickMap can round up when performing `tick += tickSpacing / 2`, there is potential for the `crossTick` to be set to the current pool price rather than the next inserted tick when a position is minted. Thus, when a trader attempts to swap, the swap is performed from the current price to the cross price which are equivalent, leading to no swap at all although liquidity is available. This goes against the core functionality of the pool where a swap should be able to be performed when liquidity is available and users can get a fill.

Consider the following scenario where LPs mint `oneForZero`:

- 1) Bob mints a position with ticks `[-100, 100]` which shifts the price to tick 100.
- 2) Alice mints a position with ticks `[-100, 100]` which calls `insertSingle` on tick 100 since liquidity is now available in the pool due to Bob's mint.
- 3) Swapper comes in to swap and the cross tick is calculated to be `TickMap.previous(tickMap, pool.tickAtPrice, cache.constants.tickSpacing, true)` with `tickAtPrice` being 100.
- 4) Due to `roundUp` being true, the tick in `Ticks.previous` becomes tick 105. The previous tick from tick 105 is now tick 100.
- 5) Because the `crossTick` is tick 100 rather than -100, no swap occurs as no ticks are crossed.

## Recommendation

Do not round up when calculating the `crossTick`.

## Resolution

Poolshark Team: The issue was resolved in commit [27c03bb](#).

# TK-2 | Liquidity Underflow Due To Tick Rounding

Category	Severity	Commit	Location	Status
Rounding	● Critical	<a href="#">5dc6b2b2b74f21c8f18a72e448ff75bf8acd84fc</a>	Ticks.sol: 365	Resolved

## Description [PoC](#)

During `MintCall.perform`, liquidity is unlocked in the swap pool in the case there is no liquidity. When the `pool.tickAtPrice` is negative, the tick rounds up when fetching the `TickMap.next()` tick. However rounding up allows the pool to skip over a tick with nonzero `liquidityDelta`.

Consequently, the `ticks[pool.tickAtPrice].liquidityDelta` can be negative with larger magnitude than the current liquidity, which will lead to silent underflow when casting the `int128` value to a `uint128` value `uint128(ticks[pool.tickAtPrice].liquidityDelta)`.

As a result, the liquidity of the swap pool becomes severely inflated due to underflow which leads to significant loss of funds for users as traders are able to swap near infinite amounts at the current pool price.

## Recommendation

Round back in `TickMap.next()` when the tick is negative.

## Resolution

Poolshark Team: The recommendation was implemented in commit [614db68](#).

# TK-3 | Cross Tick Skips Half Ticks

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">614db68cec6171fa03d41c5dde73ecdc69ee5f91</a>	Ticks.sol: 134-136	Resolved

## Description [PoC](#)

It is possible to enter the case if `(cache.amountLeft < amountMax)` during a swap, but have the `newPrice` be represented by a tick that is lower than the `cache.crossTick`. The tick of the `newPrice` can be less than the `cache.crossTick` because when `Ticks.insertSingle` is called, the `priceAt` attribute is set onto a rounded tick for the current pool price, e.g. current pool price at tick 4 is saved on the rounded tick 5.

Consequently, the tick at the `newPrice` may jump over a tick where liquidity is stashed (`liquidityDelta > 0`) during a swap. Because no cross is performed when inside `if (cache.amountLeft < amountMax)`, the liquidity is never activated.

Since the activated pool liquidity is smaller than it should be, `pool.liquidity` underflows when a future cross occurs and a tick with `liquidityDelta < 0` is crossed. This breaks a key invariant that `pool.liquidity` should never underflow, disrupting a all `LimitPool` operations.

## Recommendation

In the case that the tick at the `newPrice` is smaller/larger than the `cache.crossTick` (jumped over a tick where liquidity is stashed) in the `zeroForOne/oneForZero` case, set `cross = True` so that the stashed liquidity delta does get activated.

## Resolution

Poolshark Team: The recommendation was implemented in commit [89cce40](#).

# LMP-1 | Lack Of Access Restriction For Initialize Function

Category	Severity	Commit	Location	Status
Access Control	● Critical	<a href="#">48eb0f9ba400ff328ade4690ff1cbe37b568e6e2</a>	LimitPool.sol: 40	Resolved

## Description

The initialize function on the LimitPool has unrestricted access and can be called after initialization has already occurred.

## Recommendation

Add an onlyInitializer modifier to the initialize function so that the it cannot be called after initialization.

## Resolution

Poolshark Team: The recommendation was implemented in commit [80501d6](#).

# TK-4 | Users Can Maliciously Claim At The Current Pool Price

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">add17e0b9f5cf58ea9b55d148c76aa0090081147</a>	Ticks.sol: 449-470	Resolved

## Description [PoC](#)

When a new tick is inserted in `Positions.add` during a position mint the new tick is initialized with an epoch of 0. However the claim validation logic in `Claims.validate` relies on the epoch of the next initialized tick to determine whether or not the user should be able to claim at the supplied claim tick.

When a new position is created and a new upper and/or lower tick is created, the new tick(s) will invalidate the `Claims.validate` logic as any user can now claim at a tick that is directly previous to a newly initialized tick.

Therefore users are able to claim at the current pool price even when it is not the furthest claim tick they should be claiming at. When a user claims at the current pool price and burns liquidity, that liquidity will be removed from the `pool.liquidity` value.

However a user's position can have their liquidity stashed at a higher tick upon undercutting and therefore not have active liquidity when they are claiming at the current pool price.

This leads to users removing liquidity from the active `pool.liquidity` that should not have been and invalidates the pool accounting system leading to locked positions among other catastrophic consequences.

## Recommendation

When a new tick is inserted during `Positions.add`, do not initialize it with an epoch of 0. Instead initialize new ticks with the same epoch as the tick further along in the `zeroForOne` or `!zeroForOne` direction.

## Resolution

Poolshark Team: The recommendation was implemented in commit [8a6c3bf](#).

# TK-5 | Half Tick Liquidity Never Unlocked

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">7d27ce7b72b79ec0576dfb1655ac91f032ad8355</a>	Ticks.sol: 427	Resolved

## Description [PoC](#)

When an undercut is performed, the pool’s liquidity is stashed on a half tick with the current price.

It is crucial when swapping that this stashed liquidity is kicked into the pool, otherwise the negative `liquidityDelta` on the end tick of a position will exceed the `pool.liquidity`, causing underflow. However, it is possible for an iteration of a swap to skip a half tick where liquidity is stashed as the `cache.crossTick` can jump to the `limitTickAhead` in `_iterate()`.

For example, the `cache.crossTick` may equal 5 after:

```
(cache.crossTick,) = TickMap.roundHalf(cache.crossTick, cache.constants, cache.price);
```

But the `limitTickAhead` may be 0 after:

```
int24 limitTickAhead = TickMap.previous(limitTickMap, cache.crossTick,
cache.constants.tickSpacing, inclusive);

cache.crossTick = limitTickAhead;
```

Therefore, the tick 5 stashed liquidity is never activated since it isn’t crossed.

## Recommendation

Add the liquidity on the half tick before going to the tick ahead and then clear the tick’s liquidity delta.

## Resolution

Poolshark Team: The recommendation was implemented in commit [b633427](#).



# TK-6 | pool.price Not Updated In Ticks.unlock

Category	Severity	Commit	Location	Status
Logical Error	● Critical	<a href="#">609f5b3024a695aae4bb0ab395555959dfef4ce9</a>	Ticks.sol: 349	Resolved

## Description [PoC](#)

In the Ticks.unlock function, the ticks[pool.tickAtPrice] is zeroed out before the following logic is executed to update the pool.price:

```
uint160 priceAt = ticks[pool.tickAtPrice].priceAt;
if (priceAt > 0) {
    pool.price = priceAt;
    pool.tickAtPrice = ConstantProduct.getTickAtPrice(priceAt, cache.constants);
}
```

Since the ticks[pool.tickAtPrice] is always zeroed out before this logic, the if (priceAt > 0) case will never be entered and the pool.price will never be updated. As a result, the pool.tickAtPrice will not agree with the pool.price. Ultimately this will cause the amountMax to be significantly larger than it should be in the quoteSingle call during a swap, as it relies on the pool.price and the crossPrice which relies on the tickAtPrice.

## Recommendation

Zero out the ticks[pool.tickAtPrice] after performing the pool.price update logic which depends on the ticks[pool.tickAtPrice].

## Resolution

Poolshark Team: The recommendation was implemented in commit [1bf9eaa](#).

## TK-7 | Swaps Bricked Due To Malicious Position

Category	Severity	Commit	Location	Status
Logical Error	● High	<a href="#">91b0f3dc0e684f69825f3f8f08a3b15e875c1829</a>	Ticks.sol: 242, 243	Resolved

### **Description**

In the `quoteSingle` function, a swap cannot occur if the `pool.price` becomes `cache.constants.bounds.min` or `cache.constants.bounds.max`. Therefore a user can create a `zeroForOne` position with minimal liquidity and a lower tick of `cache.constants.bounds.min` to completely brick the pool and halt all `!zeroForOne` swaps from occurring.

This renders the pool useless, an attacker can exercise this on every Poolshark's `LimitPool` to completely shut down the protocol. Pools of the same tokens and `tickSpacing` cannot be re-deployed as they are already registered under the same key.

### **Recommendation**

Check the `pool.price` against the `cache.constants.bounds.min` and `cache.constants.bounds.max` dependent on the swap direction. If a swap is `zeroForOne`, the swap should early return if the price is `cache.constants.bounds.min` and if a swap is `!zeroForOne`, the swap should early return if the price is `cache.constants.bounds.max`.

### **Resolution**

Poolshark Team: The recommendation was implemented in commit [5dc6b2b](#).

# CLAIMS-3 | Position Resized To Half Tick

Category	Severity	Commit	Location	Status
Logical Error	● High	<a href="#">91b0f3dc0e684f69825f3f8f08a3b15e875c1829</a>	Claims.sol: 15	Resolved

## Description [PoC](#)

When a user claims for their partially filled position, they are able to claim at a half tick that is not an even multiple of their `tickSpacing`. This allows users to claim fills dependent on the stashed `priceAt` on the half tick, however it also results in the position getting resized to a start tick that happens to be that half tick.

Positions with a half tick as a boundary break a fundamental invariant of the protocol and potentially lead to severe issues and manipulation.

## Recommendation

Do not resize positions to the boundary of a half tick, instead round the new boundary tick back to the previous full tick.

## Resolution

Poolshark Team: The recommendation was implemented in commit [0e40b90](#).

# LMP-2 | Unclaimable Fees

Category	Severity	Commit	Location	Status
Logical Error	● High	<a href="#">6ab8d15c6ef9981291224c615b73f118851e6415</a>	LimitPool.sol: 187	Resolved

## Description

The fees function never sets token0Fees and token1Fees variables. Therefore, the owner will never be able to collect fees.

## Recommendation

Set token0Fees and token1Fees prior to zeroing out the protocol fees.

## Resolution

Poolshark Team: The recommendation was implemented in commit [b7ebe31](#).

# TMAP-2 | Incorrect Tick Rounding

Category	Severity	Commit	Location	Status
Rounding	● High	<a href="#">e592dd31338bef73878ccc95ceb9c9821d0db9e8</a>	TickMap.sol: 284, 303	Resolved

## Description

In the `roundAheadWithPrice` function, if `zeroForOne` and the `roundedTick` is negative, the `tickSpacing` is subtracted from the rounded tick.

Otherwise if `!zeroForOne` and the `roundedTick` is positive, the `tickSpacing` is added to the rounded tick. Therefore for the `zeroForOne` case a positive `roundedTick` is rounded down and not adjusted upwards, meanwhile a negative `roundedTick` is adjusted to be more negative. Both of these are rounding back rather than rounding ahead for a `zeroForOne` position.

For the `!zeroForOne` case a positive `roundedTick` is adjusted upwards, meanwhile a negative `roundedTick` is rounded to be less negative and is not adjusted to be more negative. Both of these are rounding back rather than rounding ahead for a `!zeroForOne` position.

Ultimately this results in the beginning of a position being resized to the `roundedBack` tick rather than the `roundedAhead` tick which unexpectedly alters the user’s overall execution price and unexpectedly sets the latest `swapEpoch` on the `roundedBack` tick.

Additionally, the `roundAhead` function implements incorrect tick rounding where for `zeroForOne` cases where the `roundedTick` is negative it is rounded up twice. First the magnitude of the negative tick is reduced with rounding and then it is further adjusted up by the `tickSpacing`. On the other hand, positive `roundedTicks` are rounded down and not adjusted up. The inverses are true for the `!zeroForOne` case.

## Recommendation

Use the following cases to accurately round ahead:

```
if (zeroForOne && (roundedTick > 0 || (roundedTick == 0 && tick > 0))) roundedTick += tickSpacing;
else if (!zeroForOne && (roundedTick < 0 || (roundedTick == 0 && tick < 0))) roundedTick -= tickSpacing;
```

## Resolution

Poolshark Team: The recommendation was implemented in commit [4666f91](#).

# GLOBAL-2 | Odd Tick Spacing Should Not Be Used

Category	Severity	Commit	Location	Status
Configuration	● High	<a href="#">5369f670c79ff8f3d0b8e010cbb797f63d24d45c</a>	Global	Resolved

## Description [PoC](#)

When an odd tick spacing is used in a `LimitPool`, tick rounding errors can cause swaps to have access to more liquidity at the current market price than they should.

Specifically in the `TickMap.previous` or `TickMap.next` functions the previous or next tick may be errantly rounded in the `_tick` function such that it yields a tick that was never set in the `TickMap`. This affects many areas of the protocol but invalidates the accounting system and leads to direct loss of funds when swappers are able to swap for an extra tick length using the same liquidity.

## Recommendation

Do not allow an odd `tickSpacing` to be used as it is incompatible with the system.

## Resolution

Poolshark Team: Odd tick spacings are now disallowed in the `LimitPoolManager` contract.

# POS-1 | Yield Can Be Stolen From Liquidity Providers

Category	Severity	Commit	Location	Status
Logical Error	● High	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Global	Resolved

## Description [PoC](#)

When computing the value of fees for a position the `priceLower` or `priceUpper` is used as the `currentPrice`, however the `priceLower` or `priceUpper` will rarely be accurate to the current price.

```
cache.liquidityOnPosition = ConstantProduct.getLiquidityForAmounts(
    cache.priceLower,
    cache.priceUpper,
    position.amount0 > 0 ? cache.priceLower : cache.priceUpper,
    position.amount1,
    position.amount0
)
```

A malicious actor can leverage this inaccuracy to mint a position where the fees are undervalued from the `liquidityOnPosition` calculation and subsequently burn to receive the full value of the fees with the calculation in the `remove` function.

```
params.amount = uint128(uint256(params.amount) * cache.totalSupply /
(uint256(position.liquidity - params.amount) + cache.liquidityOnPosition));
/// @dev - if there are fees on the position we mint less positionToken
```

## Recommendation

Convert the position accounting logic to an ERC721 implementation rather than an ERC1155 implementation to avoid unnecessary complexity with fee valuation and potential manipulation.

## Resolution

Poolshark Team: The recommendation was implemented in commit [3356f37](#).

# POS-2 | liquidityGlobal Not Decrementated

Category	Severity	Commit	Location	Status
Logical Error	● High	<a href="#">91b0f3dc0e684f69825f3f8f08a3b15e875c1829</a>	Positions.sol: 374, 378	Resolved

## Description

When `params.amount == 0` the `pool.liquidityGlobal` is decremented when the `params.claim != params.lower` and `params.claim == params.lower` for `zeroForOne` and when the `params.claim != params.upper` and `params.claim == params.upper` for `!zeroForOne`.

These conditions are unsatisfiable, therefore the `pool.liquidityGlobal` will not be decremented for positions that are fully filled and ought to have their liquidity removed from `pool.liquidityGlobal`. This way an attacker can continuously open positions and remove them until the `pool.liquidityGlobal` reaches the maximum and users are unable to mint positions rendering the pool useless.

## Recommendation

Appropriately decrement the `pool.liquidityGlobal` when users are claiming at their end tick with `params.amount == 0`.

## Resolution

Poolshark Team: The recommendation was implemented in commit [f0d52ad](#).



# CLAIMS-4 | Position Overwritten At Claim Tick

Category	Severity	Commit	Location	Status
Logical Error	● High	<a href="#">1c76102f06533900b4252e71849d1ef5f5d907f1</a>	Claims.sol: 115-123	Resolved

## Description [PoC](#)

It is possible for a user’s position to get overwritten at the claim tick because the positions mapping is accessed with the wrong ticks when performing claim tick validation.

For a zeroForOne position, the new position should span from the claim tick to the upper tick. For a !zeroForOne position, the new position should span from the lower tick to the claim tick. However, that is not how the validation is checking the user’s position.

```
// prevent position overwriting at claim tick
if (params.zeroForOne) {
    if (positions[params.owner][params.lower][params.claim].liquidity > 0) {
        require (false, string.concat('UpdatePositionFirstAt(', String.from(params.lower), ',
            ', String.from(params.claim), ')'));
    }
} else {
    if (positions[params.owner][params.claim][params.upper].liquidity > 0) {
        require (false, string.concat('UpdatePositionFirstAt(', String.from(params.lower), ',
            ', String.from(params.claim), ')'));
    }
}
```

This can lead to a trader losing their funds, because any deltas for another position they have may be overwritten when burning one of their positions.

## Recommendation

Modify the validation such that zeroForOne checks the position spanning from the params.claim to the params.upper and !zeroForOne checks the position spanning from the params.lower to the params.claim, which will prevent a claim tick that leads to an overwritten position.

## Resolution

Poolshark Team: The recommendation was implemented in commit [c9a3a42](#).

# CLAIMS-5 | Users Prevented From Burning

Category	Severity	Commit	Location	Status
Logical Error	● High	<a href="#">00db8d3494769b7363007322d3b75a9975982519</a>	Claims.sol: 39, 57	Resolved

## Description

When a user's position is undercut and liquidity is stashed on a half tick, they are required to claim at that half tick. However the initial priceClaim for a half tick is assigned to the price at that half tick rather than the priceAt for the half tick. When the pool.price is ahead of the priceClaim then the params.claim will be set to the earlier full tick.

However this tick is not a valid tick to claim at for the user, since the validation will fetch the next tick, the half tick which their position is stashed on, and check the epoch and see that the half tick is a valid claim tick and therefore revert.

This prevents users from burning their liquidity when the position is in this state, a malicious actor can abuse this to prevent others from burning from their positions and keeping them trapped.

## Recommendation

Initialize the cache.priceClaim to be the priceAt for the half tick rather than the price at the half tick.

## Resolution

Poolshark Team: The recommendation was implemented in commit [0e40b90](#).

# TK-8 | exactOut Does Not Function As Expected

Category	Severity	Commit	Location	Status
Logical Error	● High	<a href="#">040812eb01c3f998b35777166051ee955d766081</a>	Ticks.sol: 138	Resolved

## Description

The amountLeft for !exactIn is increased in accordance with the swapFee in order to give the user exactly their specified amount. However the swapFee will likely not apply to the entire amountOut, as the swapFee applies only to the portion of amountOut that was a direct result of the range pool liquidity.

Therefore users specifying an amount for !exactIn will in most cases receive more than their defined amount out, which invalidates the definition of !exactIn.

## Recommendation

Compute and apply the fees to the amountIn for the !exactIn case.

## Resolution

Poolshark Team: The recommendation was implemented in commit [90fb6e9](#).

# LMP-3 | protocolFee0 Overwritten

Category	Severity	Commit	Location	Status
Logical Error	● Medium	<a href="#">e592dd31338bef73878ccc95ceb9c9821d0db9e8</a>	LimitPool.sol: 182-183	Resolved

## Description

When assigning fees with the fees function, the LimitPoolManager will provide a protocolFee0 and a protocolFee1, however the protocolFee0 will always be overwritten with the protocolFee1.

```
globalState.protocolFee = protocolFee0;  
globalState.protocolFee = protocolFee1;
```

Therefore the protocolFee1 will always apply instead of the protocolFee0 which will result in unexpected fees being applied.

## Recommendation

Create protocolFee0 and protocolFee1 attributes on the ILimitPoolStructs.GlobalState struct to store each protocolFee.

## Resolution

Poolshark Team: The recommendation was implemented in commit [80501d6](#).

# GLOBAL-3 | Fee-On-Transfer Tokens

Category	Severity	Commit	Location	Status
Fee-on-transfer	● Medium	<a href="#">5369f670c79ff8f3d0b8e010cbb797f63d24d45c</a>	Global	Acknowledged

## Description

The LimitPoolFactory allows for permissionless creation of a LimitPool with any tokenIn and tokenOut. Therefore a token0 or token1 with fee-on-transfer or rebase mechanisms may be supplied.

In the transferIn function there is logic to handle fee-on-transfer and rebase tokens. However in the mint call the returned value is not used in the mint process.

Therefore even though there is logic built in to support fee on transfer tokens, it is not used.

## Recommendation

Refactor the mintCall and other relevant functions to rely on the returned value from transferIn to account for fee on transfer tokens. Otherwise make it well documented that fee-on-transfer and rebase tokens are not compatible with the system.

## Resolution

Poolshark Team: Acknowledged.

# POS-3 | Small Prices Round Out of Range When Multiplied

Category	Severity	Commit	Location	Status
Rounding	● Medium	<a href="#">91b0f3dc0e684f69825f3f8f08a3b15e875c1829</a>	Positions.sol: 64	Acknowledged

## Description

When minting a position with a lower tick at or below tick -665460, mints will begin to revert with the `priceOutOfBounds` error. This is because the `getLiquidityForAmounts` function returns 0 when the product of `priceLower` and `priceUpper` is less than Q96.

When `liquidityMinted` is zero it causes the `priceLimit` to be 0 as well. Therefore failing the validation in the `getTickAtPrice` function as 0 is less than the price limit.

```
if (price < constants.bounds.min || price >= constants.bounds.max)
    require (false, 'PriceOutOfBounds()');
```

If the price of a pool were in this range user’s ability to mint positions would be limited as they would have to increase the range well beyond market price to mint successfully.

## Recommendation

Consider implementing a solution for the `getLiquidityForAmounts` function when the product of the `priceLower` and `priceUpper` are less than Q96. Otherwise ensure this behavior is clearly documented for traders and users deploying pools.

## Resolution

Poolshark Team: Acknowledged.

# POS-4 | Resizes Without Swaps Are Too Severe

Category	Severity	Commit	Location	Status
Logical Error	● Medium	<a href="#">5369f670c79ff8f3d0b8e010cbb797f63d24d45c</a>	Positions.sol: 61	Resolved

## Description

When positions are resized and the priceLimit is not past the current pool price the resulting position is still resized to the priceLimit, rather than the current market price.

This is unexpected for the user and does not allow their position to immediately begin filling as price moves in their direction.

## Recommendation

Add the following after fetching the cache.priceLimit in order to resize the user to the current market price in the event that a swap cannot occur.

```
if (ConstantProduct.withinBounds(cache.swapPool.price, cache.constants) &&
(params.zeroForOne ? cache.priceLimit > cache.swapPool.price : cache.priceLimit <
cache.swapPool.price)) { cache.priceLimit = cache.swapPool.price; }
```

## Resolution

Poolshark Team: The recommendation was implemented in commit [501f609](#).

# POS-5 | Misleading liquidityBurned emitted

Category	Severity	Commit	Location	Status
Events	● Medium	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Positions.sol: 375, 383	Resolved

## Description

The Burn functionality allows users to burn a fully filled position with a nonzero `params.amount`. In this case the regular burn `params.amount > 0` logic is entered and the `position.liquidity` is decremented by the `params.amount`.

Subsequently the position removal case is entered and the `params.amount` is updated to be the `position.liquidity` after the `position.liquidity` has been reduced by the original `params.amount`.

The `params.amount` is then used to emit the `BurnLimit` event as the `liquidityBurned` parameter.

Therefore a user can mislead other parties relying on the `BurnLimit` event by providing a nonzero `params.amount` when burning their fully filled position. In the worst case this event may emit a `liquidityBurned` of 0, while the entire original `position.liquidity` was actually burned.

## Recommendation

Allowing users to provide a nonzero `params.amount` when they are burning a fully filled position introduces more avenues for exploitation and inconsistency. Do not allow users to pass a nonzero `params.amount` when they are burning their fully filled position, or auto update the amount to be 0 in `Claims.validate`.

## Resolution

Poolshark Team: The recommendation was implemented in commit [0e40b90](#).



# POS-6 | Position Minted With 0 Liquidity

Category	Severity	Commit	Location	Status
Logical Error	● Medium	<a href="#">00db8d3494769b7363007322d3b75a9975982519</a>	Positions.sol: 138-145	Resolved

## Description [PoC](#)

It is possible to insert a position into the positions mapping with 0 liquidity. When minting a position, the initial `cache.liquidityMinted` in `Positions.resize` may be greater than 0, bypassing the `if (cache.liquidityMinted == 0) require (false, 'PositionLiquidityZero()')` check.

However, when the `cache.liquidityMinted` is recalculated in the below snippet, it can become 0 after accounting for the swap.

```
if (params.amount > 0 && params.lower < params.upper)
    cache.liquidityMinted = ConstantProduct.getLiquidityForAmounts(
        cache.priceLower,
        cache.priceUpper,
        params.zeroForOne ? cache.priceLower : cache.priceUpper,
        params.zeroForOne ? 0 : uint256(params.amount),
        params.zeroForOne ? uint256(params.amount) : 0
    );
```

This causes the user slight loss as any leftover `params.amount` will be unclaimable.

## Recommendation

Check if the `cache.liquidityMinted == 0` on recalculation, and revert if so.

## Resolution

Poolshark Team: The recommendation was implemented in commit [5dc6b2b](#).

# FMATH-1 | Inaccurate Fee On Event

Category	Severity	Commit	Location	Status
Events	● Medium	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	FeeMath.sol: 98-100	Resolved

## Description

The `cache.feeAmount` is increased by the `locals.feeAmount` after the `locals.feeAmount` is decreased by the `locals.protocolFeesAccrued`.

```
locals.feeAmount -= locals.protocolFeesAccrued;  
// add to total fees paid for swap  
cache.feeAmount += locals.feeAmount.toUint128();
```

As a result, the `cache.feeAmount` emitted in the `Swap` event does not truly represent how much the user paid in fees after swapping, since the protocol fees are not accounted for.

## Recommendation

Include the protocol fees the user paid as part of the `cache.feeAmount`.

## Resolution

Poolshark Team: The recommendation was implemented in commit [c312551](#).

# TMAP-3 | Inclusive Skips Half Tick On Next

Category	Severity	Commit	Location	Status
Logical Error	● Medium	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	TickMap.sol: 133	Resolved

## Description

In the case that the `tickAtPrice` is past the half tick, calling `next()` with `inclusive=True` does not return the the half tick when it is set. This could potentially lead to problems with liquidity not being activated which causes further negative consequences such as underflow, swap failure, among other things.

## Recommendation

Amend the inclusive logic such that a tick that is past the half tick but not yet at the full tick rounds back to the half tick by reducing the magnitude of the tick by half of the `tickSpacing`.

## Resolution

Poolshark Team: The recommendation was implemented in commit [3356f37](#).

# BCALL-2 | State Saved After Token Transfer

Category	Severity	Commit	Location	Status
Reentrancy	● Medium	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	BurnLimitCall.sol: 79	Resolved

## Description

In the perform function the Collect.burnLimit function which transfers tokens is executed before saving the state of the position.

Poolshark allows the creating of a LimitPool with any tokenIn and tokenOut, some of these tokens will have callback capabilities. In the event that tokens with callbacks are used the state of the position should be saved before executing any token transfers, this way avoiding read-only reentrancy risks for systems interacting with Poolshark.

## Recommendation

Save the state of the position before transferring out tokens to the receiver.

## Resolution

Poolshark Team: The recommendation was implemented in commit [77f9c26](#).

# LMP-4 | Quote and Snapshot Read-only Reentrancy Risk

Category	Severity	Commit	Location	Status
Reentrancy	● Medium	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Global	Resolved

## Description

The quote and snapshot view functions lack reentrancy protection and therefore allow a malicious user to execute a flashloan swap from the LimitPool and manipulate the output of quote/snapshot to exploit systems interacting with the LimitPool and relying on these view functions.

## Recommendation

Add reentrancy protections to the quote and snapshot functions such that these functions revert if the system has already been entered during the transaction.

## Resolution

Poolshark Team: The recommendation was implemented in commit [9458d42](#).

# RPEI-1 | Missing ERC1155 Support Validation

Category	Severity	Commit	Location	Status
Validation	● Medium	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	RangePoolERC1155.sol: 78	Resolved

## Description

The mintFungible function lacks a checkERC1155Support modifier to validate that the \_account address which will receive the minted tokens can handle the ERC1155 tokens.

## Recommendation

Add a checkERC1155Support(\_account) modifier to the mintFungible function.

## Resolution

Poolshark Team: The recommendation was implemented in commit [97d3d50](#).

# PROU-2 | Lacking Slippage Controls

Category	Severity	Commit	Location	Status
Slippage	● Medium	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	PoolRouter.sol	Acknowledged

## Description

The PoolRouter contract lacks any explicit slippage controls such as minimum output amount or maximum input amount. The priceLimit parameter for the LimitPool.swap function serves as user’s only form of protection from sandwich attacks, however user’s may desire additional explicit slippage controls such as minimum output or maximum input.

## Recommendation

Consider implementing minimum output and maximum input slippage controls in the PoolRouter contract.

## Resolution

Poolshark Team: Acknowledged.

# LPF-1 | Lack Of Token Validation

Category	Severity	Commit	Location	Status
Validation	● Low	<a href="#">af17ff9fde824eb83686ac6e6217512f39c5b468</a>	LimitPoolFactory.sol: 24	Resolved

## Description

When creating a new LimitPool with the createLimitPool function there is no validation that tokenIn  $\neq$  tokenOut or that neither tokenIn nor tokenOut are address(0).

Certainly pools where token1 == token0 are invalid. Additionally, pools where token0 is address(0) will attempt to use native ether via the functions in SafeTransfers.sol, however no functions are payable so clearly native ether is incompatible with the system.

## Recommendation

Add validation to check that tokenIn  $\neq$  tokenOut as well as that token0  $\neq$  address(0). Otherwise if token0 should be allowed to be address(0) and native ether is indeed meant to be compatible with the system, make the appropriate functions payable to allow native ether to be used in the system.

## Resolution

Poolshark Team: The suggested validations were implemented in the createLimitPool function.



# POS-7 | Superfluous finalTick Assignment

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">af17ff9fde824eb83686ac6e6217512f39c5b468</a>	Positions.sol: 269-273	Resolved

## Description

The finalTick is read from storage before being immediately assigned back to storage.

```
{
  // update max deltas
  ILimitPoolStructs.Tick memory finalTick =
  ticks[params.zeroForOne ? params.lower : params.upper];
  ticks[params.zeroForOne ? params.lower : params.upper] = finalTick;
}
```

## Recommendation

Remove the read and write for the finalTick as there is no net effect.

## Resolution

Poolshark Team: The recommendation was implemented in commit [3682ae9](#).

# POS-8 | Superfluous Else Case

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">af17ff9fde824eb83686ac6e6217512f39c5b468</a>	Positions.sol: 225	Resolved

## Description

In the `remove` function, on line 225 an `if` case performs validation on the position's liquidity and reverts if the case is entered.

The following logic of the function is nested within an `else` case, however this `else` case is unnecessary as the contents of the `if` case will always revert.

## Recommendation

Move the subsequent logic outside of the `else` case to improve code readability and style.

## Resolution

Poolshark Team: The recommendation was implemented in commit [3682ae9](#).

# GLOBAL-4 | Unused Q96 Constant

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">af17ff9fde824eb83686ac6e6217512f39c5b468</a>	Global	Resolved

## Description

In the Positions.sol and Ticks.sol files there is a Q96 constant, but it is never used.

## Recommendation

Remove the Q96 constant from these files.

## Resolution

Poolshark Team: The recommendation was implemented in commit [3682ae9](#).

# TK-9 | Typo

Category	Severity	Commit	Location	Status
Typo	● Low	<a href="#">af17ff9fde824eb83686ac6e6217512f39c5b468</a>	Ticks.sol: 461	Resolved

## Description

The following comment contains a typo:

would be smart to protect against the case of epochs crossing

## Recommendation

would be smart to protect against the case of epochs crossing

## Resolution

Poolshark Team: The recommendation was implemented in commit [3682ae9](#).

# TK-10 | Unnecessary Ternary Operator

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">af17ff9fde824eb83686ac6e6217512f39c5b468</a>	Ticks.sol: 426, 439	Resolved

## Description

In the Ticks.insert function, ternary operators are used as conditionals where one case is always true. However these ternaries can be simplified in the following way:

```
params.zeroForOne ? cache.priceLower > cache.pool.price : true
```

→

```
!params.zeroForOne || cache.priceLower > cache.pool.price
```

```
params.zeroForOne ? true : cache.priceUpper < cache.pool.price
```

→

```
params.zeroForOne || cache.priceLower > cache.pool.price.
```

## Recommendation

Implement the above suggested simplifications.

## Resolution

Poolshark Team: The recommendation was implemented in commit [91b0f3d](#).

# LPF-2 | Inefficient Token Assignment

Category	Severity	Commit	Location	Status
Optimization	● Low	<a href="#">af17ff9fde824eb83686ac6e6217512f39c5b468</a>	LimitPoolFactory.sol: 32	Resolved

## Description

When assigning the token0 and token1 for a new limit pool, two ternary operators are used to determine each token. However a single ternary operator can be used like so:

```
(address token0, address token1) = tokenIn < tokenOut ? (tokenIn, tokenOut) : (tokenIn, tokenOut)
```

## Recommendation

Implement the above suggested optimization.

## Resolution

Poolshark Team: The recommendation was implemented in commit [91b0f3d](#).

# GLOBAL-5 | Superfluous GlobalState Storage Variable

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">48eb0f9ba400ff328ade4690ff1cbe37b568e6e2</a>	Global	Resolved

## Description

Throughout the codebase the GlobalState globalState storage variable is used however the GlobalState struct only contains an unlocked variable to facilitate reentrancy locks. However the reentrancy lock currently implemented by the system is a more error prone and less efficient version of OpenZeppelin’s ReentrancyGuard.

## Recommendation

Remove the globalState variable and use OpenZeppelin’s ReentrancyGuard.

## Resolution

Poolshark Team: The GlobalState storage variable is now used for more than a reentrancy lock.

# POS-9 | Inconsistent mintPercent Decimals

Category	Severity	Commit	Location	Status
Consistency	● Low	<a href="#">add17e0b9f5cf58ea9b55d148c76aa0090081147</a>	Positions.sol: 47	Acknowledged

## Description

In the Positions.resize function, the params.mintPercent is treated as a percentage with 1e28 decimals, however this is inconsistent with the decimals of 1e38 used for liquidity percent conversions in the \_convert function.

## Recommendation

Standardize on either 1e28 or 1e38 for percentage decimals for consistency.

## Resolution

Poolshark Team: Acknowledged.



# LMP-5 | Lacking Zero Address Checks

Category	Severity	Commit	Location	Status
Validation	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	LimitPool.sol: 73, 128	Acknowledged

## Description

There is no check for `params.to == address(0)` when calling the `mint` or `swap` functions in `LimitPool`.

## Recommendation

Implement the following check in the `mint` and `swap` functions:  
`if (params.to == address(0)) revert CollectToZeroAddress();`

## Resolution

Poolshark Team: Acknowledged.

# LMP-6 | Typo

Category	Severity	Commit	Location	Status
Typo	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	LimitPool.sol	Acknowledged

## Description

The canonicalOnly modifier is misspelled as canoncialOnly.

## Recommendation

Replace all instances of canoncialOnly with canonicalOnly.

## Resolution

Poolshark Team: Acknowledged.

# EMAP-1 | Redundant \_tick Function

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	EpochMap.sol: 97	Acknowledged

## Description

The `_tick` function is implemented in both the `EpochMap` and the `TickMap` files. Only the `_tick` implementation in the `TickMap` is used, therefore the implementation in the `EpochMap` can be removed.

## Recommendation

Remove the `_tick` function from the `TickMap`.

## Resolution

Poolshark Team: Acknowledged.

# RLIB-1 | Superfluous RebaseLibrary

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">00a7518c0528dd4624070d424254fd23191b75e4</a>	Rebase.sol	Acknowledged

## Description

Throughout the codebase the `RebaseLibrary.sol` file is not used.

## Recommendation

Remove the unnecessary library.

## Resolution

Poolshark Team: Acknowledged.

# CLAIMS-6 | Superfluous claimTick Assignments

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Claims.sol: 15	Acknowledged

## Description

The `cache.claimTick` is assigned to in several cases throughout the `Claims.validate` function, however the `cache.claimTick` is never referenced after this assignment.

## Recommendation

Remove the assignments in the `Claims.validate` function.

## Resolution

Poolshark Team: Acknowledged.

# TK-11 | Outdated Comments

Category	Severity	Commit	Location	Status
Documentation	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Ticks.sol: 535, 551	Acknowledged

## Description

The comment 0 -> 1 positions price moves up so nextFullTick is lesser is not accurate on line 535 as the relevant tick is the previousFullTick rather than the nextFullTick.

The comment 0 -> 1 positions price moves up so nextFullTick is lesser is not accurate on line 551 as this is the !zeroForOne case.

## Recommendation

Update the comments in the insertSingle function.

## Resolution

Poolshark Team: Acknowledged.

# CLAIMS-7 | Superfluous claimTickEpoch Assignments

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Claims.sol: 43, 61	Acknowledged

## Description

The `claimTickEpoch` is assigned a value inside of the inner `if` cases on lines 43 and 61 but then re-assigned to the same exact value right after.

## Recommendation

Remove the assignments to `claimTickEpoch` on these lines as they have no effect.

## Resolution

Poolshark Team: Acknowledged.

# SFST-1 | Unused Checks Library

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	SafeState.sol	Acknowledged

## Description

The Checks library includes save and balance functions which may be useful throughout the codebase, however this library is never used.

## Recommendation

Either implement usage of the Checks library or remove it from the codebase.

## Resolution

Poolshark Team: Acknowledged.



# POS-10 | Unreachable Code

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Positions.sol: 391	Acknowledged

## Description

The if statement beginning on line 391 is unreachable as it requires the `params.claim != lower` as well as `params.claim == lower` which is unsatisfiable.

## Recommendation

Remove this if statement as it is unreachable and unnecessary.

## Resolution

Poolshark Team: Acknowledged.

# CLAIMS-8 | Superfluous Early Return Logic

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">add17e0b9f5cf58ea9b55d148c76aa0090081147</a>	Claims.sol	Acknowledged

## Description

The early return logic in `Claims.validate` is unreachable. The conditions that satisfy the early return case in `Claims.validate` constitute calling `Positions.remove` instead.

To enter update (for `zeroForOne`):

```
cache.position.claimPriceLast != 0 || params.claim != params.lower || epochLower > positionEpoch
```

To enter the early return case inside of update (for `zeroForOne`):

```
claimPriceLast == 0 && params.claim == params.lower && epochLower <= positionEpoch
```

Never will these two conditions both be satisfiable, therefore this early return case can never be reached.

## Recommendation

Remove all of the early return logic as it is unreachable.

## Resolution

Poolshark Team: Acknowledged.

# POS-11 | Superfluous Position Write

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Positions.sol: 281	Acknowledged

## Description

There is a positions mapping write in the Positions.remove function, however it will always write to the same lower and upper tick boundaries as the subsequent write in the BurnCall.perform function.

## Recommendation

Remove the extraneous positions write in the Positions.remove function, additionally add a check in the Positions.remove function that the claim is always exactly the params.lower for zeroForOne and params.upper for !zeroForOne, to be explicitly safe.

## Resolution

Poolshark Team: Acknowledged.

# LMPM-1 | Enabled Pool Configurations Cannot Be Disabled

Category	Severity	Commit	Location	Status
Configuration	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	LimitPoolManager.sol	Acknowledged

## Description

Once a tickSpacing or implementation is configured in the LimitPoolManager contract they cannot be disabled.

If an issue is discovered with any particular tickSpacing or implementation then it cannot be removed and pools will still be allowed to be created with that misguided configuration.

## Recommendation

Implement functions to disable a particular tickSpacing or implementation in the LimitPoolManager.

## Resolution

Poolshark Team: Acknowledged.

# POS-12 | Redundant Validation

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">6ab8d15c6ef9981291224c615b73f118851e6415</a>	Positions.sol: 248	Acknowledged

## Description

In the `remove` function, it is validated that the `params.amount` is less than or equal to `1e38`. However this validation is already performed in the `_convert` function.

## Recommendation

Remove the first instance of the validation and perform the conversion at the beginning of the `remove` function.

## Resolution

Poolshark Team: Acknowledged.

# GLOBAL-6 | Superfluous refundTo Variable

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Global	Acknowledged

## Description

There is a refundTo parameter on the MintParams struct, however it is never initialized or referenced.

## Recommendation

Either implement logic for the refundTo address or remove it from the MintParams struct.

## Resolution

Poolshark Team: Acknowledged.

# GLOBAL-7 | Floating Pragma Version

Category	Severity	Commit	Location	Status
Floating Pragma	● Low	<a href="#">6ab8d15c6ef9981291224c615b73f118851e6415</a>	Global	Resolved

## Description

The smart contracts in the project use floating version of Solidity (^0.8.13). There is a list of known bugs in Solidity versions, many of them can impact smart contract functionality in unexpected way: <https://github.com/ethereum/solidity/blob/develop/docs/bugs.json>.

Furthermore, if compiled with 0.8.20 there may be unexpected reverts when deployed as some chains still do not support PUSH0.

## Recommendation

Consider utilizing a static version.

## Resolution

Poolshark Team: The recommendation was implemented in [0dd6a82](#).

# GLOBAL-8 | For Loop Optimizations

Category	Severity	Commit	Location	Status
Optimization	● Low	<a href="#">6ab8d15c6ef9981291224c615b73f118851e6415</a>	Global	Resolved

## Description

Throughout the codebase several for loops are used without caching the length of the array they iterate over or performing an unchecked { ++i } for optimization.

## Recommendation

Consider caching the length of the array to be iterated over and incrementing the index with unchecked { ++i }.

## Resolution

Poolshark Team: The recommendation was implemented in [91ab44a](#).



# POS-13 | Superfluous Removal Logic

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Positions.sol	Acknowledged

## Description

The `EpochMap.get(params.lower, tickMap, constants) > cache.position.epochLast` and `EpochMap.get(params.upper, tickMap, constants) > cache.position.epochLast` conditions on lines 234 and 249 are unsatisfiable as these are conditions to enter `Positions.update` rather than `Positions.remove`.

In fact if these conditions were satisfiable a critical bug would arise where the `position.liquidity` is decremented from the `pool.liquidity` as well as the position's upper and lower ticks.

## Recommendation

Remove the logic from lines 232 to 261.

## Resolution

Poolshark Team: Acknowledged.

# GLOBAL-9 | Unsafe Casting

Category	Severity	Commit	Location	Status
Casting	● Low	<a href="#">5a30bbfb43f82d9c1da495a45a9fc1bcc988bfaf</a>	Global	Acknowledged

## Description

Throughout the codebase, casting operations are performed. Downcasting does not revert on overflow, therefore it would be prudent to use OpenZeppelin’s SafeCast to revert in these cases.

## Recommendation

Consider using OpenZeppelin’s SafeCast to protect against undetected overflow.

## Resolution

Poolshark Team: Acknowledged.

# GLOBAL-10 | Dust Positions Negatively Impact The Protocol

Category	Severity	Commit	Location	Status
Manipulation	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Global	Acknowledged

## Description

Malicious actors are allowed to create many small range positions with only a few wei of liquidity in order to initialize many ticks and make swaps more gas consumptive. Additionally, position's with minute liquidity and narrow ranges serve little purpose but to potentially manipulate the LimitPool in unforeseen ways.

## Recommendation

Consider implementing a minimum liquidity for position's to avoid potential manipulation or griefing, potentially based on the width of the position.

## Resolution

Poolshark Team: Acknowledged.

# CLAIMS-9 | Malicious Burn DoS

Category	Severity	Commit	Location	Status
DoS	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Claims.sol: 124, 128	Acknowledged

## Description

As a side effect of the `positions[params.owner][params.claim][params.upper].liquidity > 0` check in the claims validation, a malicious actor can front-run a user’s burn and mint a dust position for the user such that this claim validation check fails. The user would then have to burn the dust position before being able to claim their existing position.

## Recommendation

Consider removing the ability for users to mint for other users or ensure a minimum liquidity amount is implemented such that any griefing attacks pose a non-trivial loss to the actors.

## Resolution

Poolshark Team: Acknowledged.

# RNP-1 | Liquidity Overflow Check Is Inconsistent

Category	Severity	Commit	Location	Status
Validation	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	RangePositions.sol: 84	Acknowledged

## Description

The RangePositions.validate function only performs validation that the liquidityMinted is less than the max int128 value, however fails to validate that the liquidityGlobal + liquidityMinted is less than the max uint128 value.

This validation is later performed in the RangeTicks.insert function, however the disparity in validation poses risk for any future code changes.

## Recommendation

Include the liquidityGlobal validation in the RangePositions.validate function.

## Resolution

Poolshark Team: Acknowledged.

# TK-12 | Superfluous cache.liquidity Adjustments

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Ticks.sol: 451	Acknowledged

## Description

In the `_cross` function the `cache.liquidity` amount is adjusted for the `LIMIT_TICK` and `LIMIT_POOL` cross statuses. However the `cache.liquidity` is immediately overwritten in the `_iterate` function, therefore the `cache.liquidity` writes in `_cross` have no effect.

## Recommendation

Remove the unnecessary `cache.liquidity` adjustments in the `_cross` function.

## Resolution

Poolshark Team: Acknowledged.

# POS-14 | Position Unnecessarily Deleted

Category	Severity	Commit	Location	Status
Optimization	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	LimitPositions.sol	Acknowledged

## Description

The claim tick is rounded back to the earlier full tick after performing the position removal check on line 417. However in the old position would not need to be cleared when the `params.claim` is a half tick ahead of the position's lower tick.

## Recommendation

Perform `params.claim = TickMap.roundBack(params.claim, constants, params.zeroForOne, cache.priceClaim)` before checking if clearing the original position is necessary on line 417.

## Resolution

Poolshark Team: Acknowledged.

# CLAIMS-10 | Futile Burns Are Allowed

Category	Severity	Commit	Location	Status
Validation	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Claims.sol: 135	Acknowledged

## Description

The validation logic aims to prevent any burns which would result in no net change to the existing position with a `NoPositionUpdates` error.

However a burn with an amount of 0 and a `claimTick` which is just `tickSpacing/2` ahead of the position's start tick will result in no net change to the existing position yet get past the validation logic.

## Recommendation

Do not allow burns with an amount of 0 for claims where the `claimTick` rounds back to the start tick of the position.

## Resolution

Poolshark Team: Acknowledged.



# TK-13 | Unnecessary uint256 Casting

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Ticks.sol: 338	Acknowledged

## Description

Throughout the Ticks.\_quoteSingle function the cache.price variable is cast as uint256. However these variables are already uint256 variables and so therefore do not need to be cast as such.

## Recommendation

Do not cast the cache.price variable as a uint256.

## Resolution

Poolshark Team: Acknowledged.

# GLOBAL-11 | Redundant Boolean Logic In \_empty

Category	Severity	Commit	Location	Status
Optimization	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Global	Acknowledged

## Description

The `_empty` functions in `RangeTicks` and `LimitTicks` use an `if` condition to return `true` or `false`. However the `_empty` function can simply return the result of the condition rather than using an `if`.

## Recommendation

Return the result of the `liquidityAbsolute != 0` check directly.

## Resolution

Poolshark Team: Acknowledged.

# MRC-1 | Inefficient Validation

Category	Severity	Commit	Location	Status
Validation	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	MintRangeCall.sol: 41	Acknowledged

## Description

The Positions.validate call can be performed before the Positions.update as any update would be invalid if the provided lower/upper ticks are invalid.

## Recommendation

Perform Positions.validate before Positions.update.

## Resolution

Poolshark Team: Acknowledged.

# RTK-1 | Superfluous Else Case

Category	Severity	Commit	Location	Status
Superfluous Code	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	RangeTicks.sol	Acknowledged

## Description

The contents of the `else` case on lines 95 and 130 are exactly that of the above `if` case where the tick already existed.

## Recommendation

Rather than introducing another separate case with the same logic, adjust the original `if` cases to include the `lower > tickAtPrice` and `upper > tickAtPrice` cases.

## Resolution

Poolshark Team: Acknowledged.

# GLOBAL-12 | Clones Can Receive ETH But Not Withdraw It

Category	Severity	Commit	Location	Status
Trapped Ether	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	Global	Acknowledged

## Description

The clones deployed with the cloneDeterministic function implement a receive function however neither the LimitPool nor the RangePoolIERC1155 implementations have methods to withdraw Ether that may be errantly sent to the clone.

## Recommendation

Implement safety Ether withdrawal functions in case Ether is errantly transferred to the cloned contracts.

## Resolution

Poolshark Team: Acknowledged.

# BRC-1 | Superfluous Addition And Subtraction

Category	Severity	Commit	Location	Status
Optimization	● Low	<a href="#">90fb6e9bd1de8cba6950661314b8cdd3c6515247</a>	BurnRangeCall.sol: 63	Acknowledged

## Description

In the BurnRangeCall.perform function the position.amount0 and position.amount1 are decremented by the cache.amount0 and cache.amount1 after the RangePositions.remove call. The goal of this is simply to remove the burned fees amount from the position.

However the subtraction takes place after the RangePositions.remove call, where the cache.amount0 and cache.amount1 are incremented by the position liquidity removed by the burn.

For this reason the position.amount0 and position.amount1 are also increased by the removed liquidity amounts in the RangePositions.remove function.

However this unnecessary addition can be avoided by simply subtracting the cache.amount0 and cache.amount1 from the position amounts directly after the RangePositions.update call where the cache.amount0 and cache.amount1 represent exactly the burned fee amounts.

## Recommendation

Move the position.amount0 and position.amount1 subtractions directly after the RangePositions.update call.

## Resolution

Poolshark Team: Acknowledged.

# 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>