GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

POOLSHARK

Summary

Audit Firm: Guardian Audits

Client Firm: Poolshark

Prepared By: Owen Thurm, Daniel Gelfand, 0xKato

Final Report Date - May 13, 2023

Audit Summary

Poolshark engaged Guardian to review the security of their Directional AMM Cover Pool. From the 17th of March to the 14th of April, a team of 3 auditors reviewed the source code in scope. The auditing approach championed manual analysis to uncover novel exploits and verify intended behavior with ancillary verification from formal methods such as fuzzing and symbolic execution. All 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 standards set forth in the Poolshark whitepaper.

Code Quality From the 17th of March to the 14th of April, the codebase quality improved considerably. However, it is recommended to improve in-code documentation supporting NatSpec standards and to address all outstanding comments. Additionally, given the scope of changes made to the codebase, Guardian supports an independent security audit 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 Overview

Project Summary

Project Name	Poolshark
Language	Solidity
Codebase	https://github.com/poolsharks-protocol/cover
Initial Commit	0c9af263973d874c4decaeecb0ad05297cf0414d
Final Commit	19f1f868c9174a5ce8bee703e935386f55e5b228

Audit Summary

Delivery Date	May 13, 2023
Audit Methodology	Manual Review, Contract Fuzzing, Symbolic Execution

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
🔴 Critical	11	0	0	0	0	11
🟠 High	5	0	0	0	0	5
🟡 Medium	7	0	0	0	0	7
🟢 Low	13	4	0	0	0	9

Audit Scope & Methodology

Scope

ID	File	Final SHA-1 Checksum(s)
CPE	CoverPoolEvents.sol	0537e2602b675ef1349515dd3d75d48ab632cb52
CPFE	CoverPoolFactoryEvents.sol	98a2ac888a6231136b4b6132ef9fbf2cdc966877
CPME	CoverPoolManagerEvents.sol	b99461ed5841ecf53faa245a07df8885da703585
CPM	CoverPoolModifiers.sol	62cd37dcf871108cba8bc285b1fcc656bd458504
CPFS	CoverPoolFactoryStorage.sol	535b0ace07ec94feb3067e5c68b639164ed9369f
CPS	CoverPoolStorage.sol	dcddbbd74749eb4d67b77e1e2d4b682a28b4f6d0
CPFS	CoverPoolFactoryStructs.sol	31fecbd60921578acec28976ddc720d327fbf6ce
DDM	DyDxMath.sol	94ae085130ec3981d04c69f470b00aa76810ea93
FPM	FullPrecisionMath.sol	44d7ae5679bd574aeef2525023561111179ccae4e
TM	TickMath.sol	0172e649882f7c6cbaacef3d506bba5746adec45
CL	Claims.sol	86f70fdafce4af666005e8d60016813d432b43da
DT	Deltas.sol	8a2bc72b0a01ee2dca4377e6784f39b76278b4f2
EPM	EpochMap.sol	6a75bd6e2aca4aa189bb45c9f932fd424ae66b43
EP	Epochs.sol	6a75bd6e2aca4aa189bb45c9f932fd424ae66b43
PS	Positions.sol	c786b377719ddabba9e813cc8f375f5c1e3362cb
TKM	TickMap.sol	17f414d5f30eb116ad3ed3c2686dea6fae4e58bc
TK	Ticks.sol	a997686c05b2e1f69ff81d0e3807dcf0af5e230a

Audit Scope & Methodology

Scope

ID	File	Final SHA-1 Checksum(s)
CPE	CoverPoolErrors.sol	94568aa3feb5f3f26767e7345f4ea1d370fe52ba
CPM	CoverPoolManager.sol	6f30aae0fd8febe7371c1ae084a6429a5912ac3d
ST	SafeTransfers.sol	615657ec5c2884995276ef172f2cffd7fc2adea9
CP	CoverPool.sol	2cf31995d6c332993d2029fcb0ce948c6f05a12b
CPF	CoverPoolFactory.sol	55dc4c3f4057b6d02f9a4fc8c91664fd24db2641

Audit Scope & Methodology

Methodology

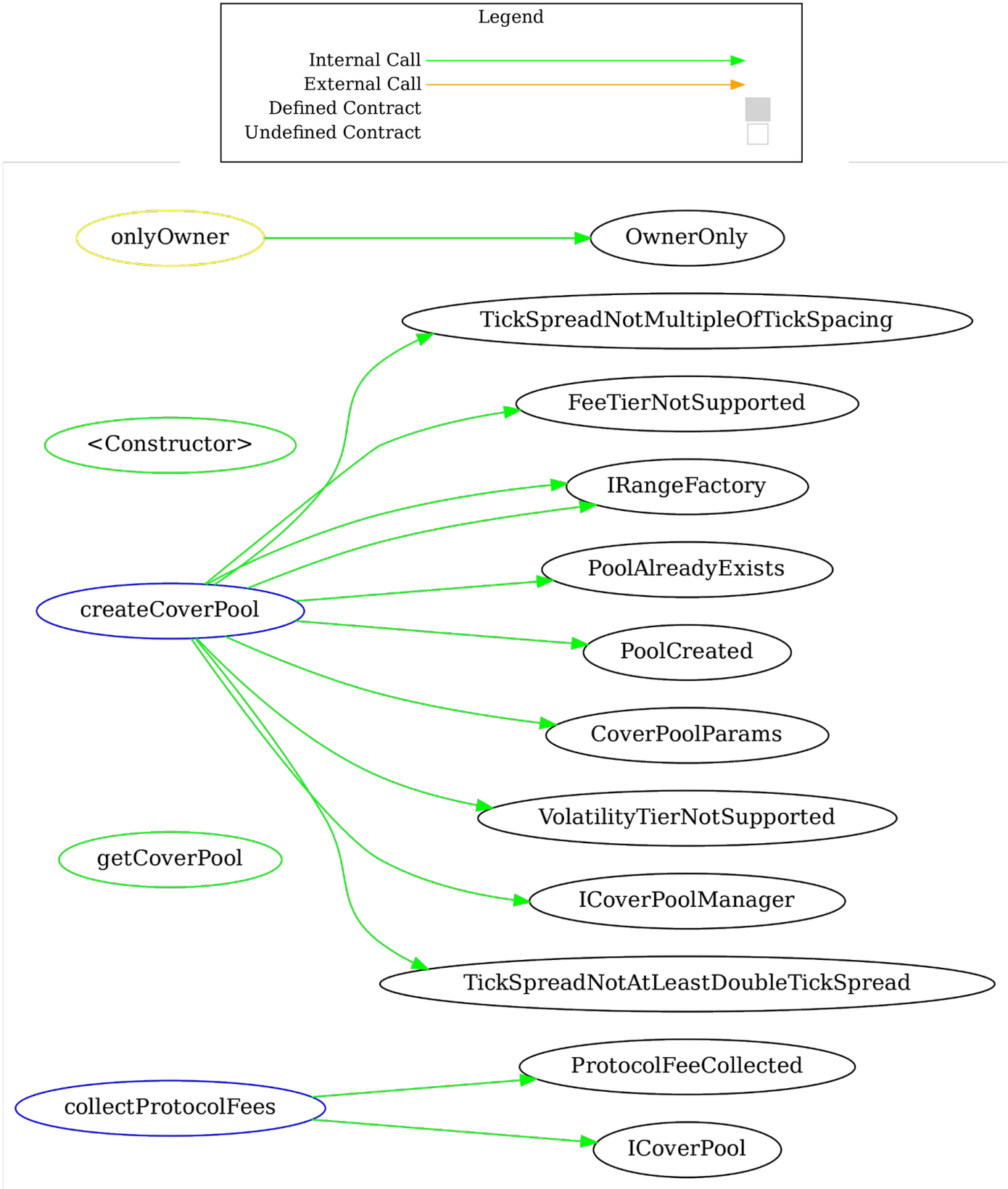
The auditing process pays special attention to the following considerations:

- 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 behavior.
- Symbolic Execution for verification of intended behavior.

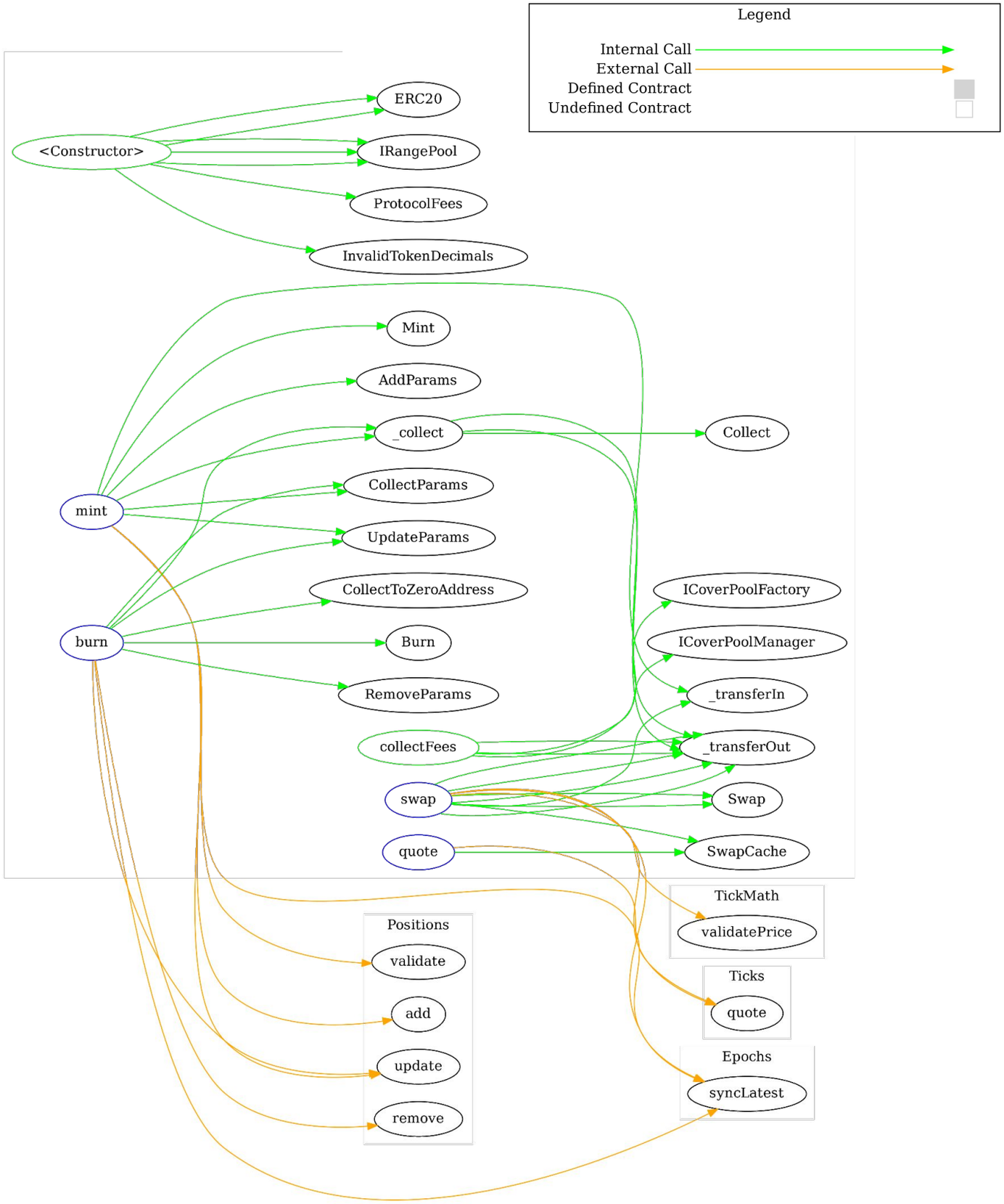
Vulnerability Classifications

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.

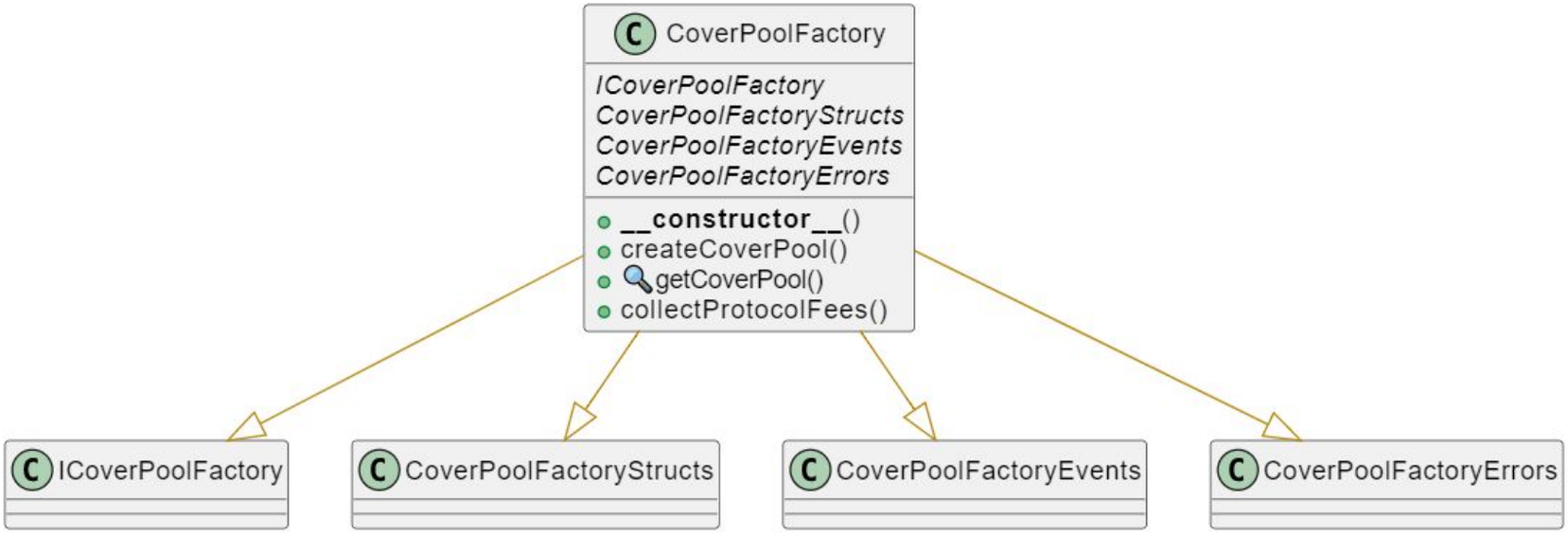
Call Graph - CoverPoolFactory



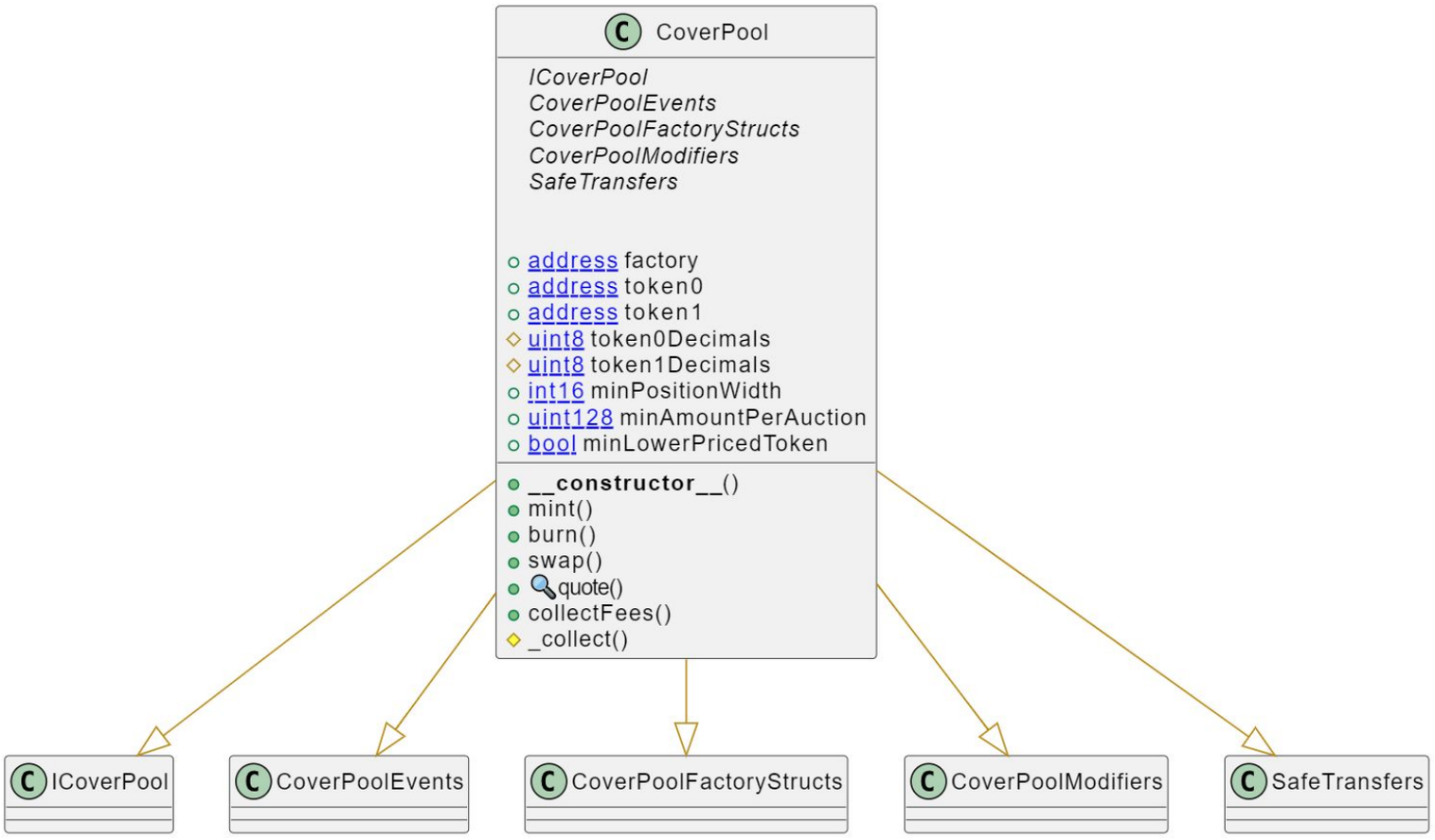
Call Graph - CoverPool

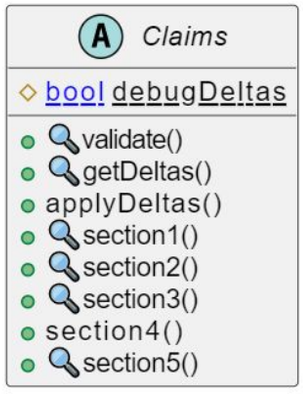


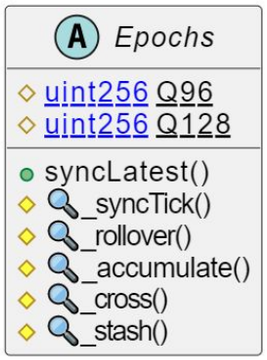
UML - CoverPoolFactory

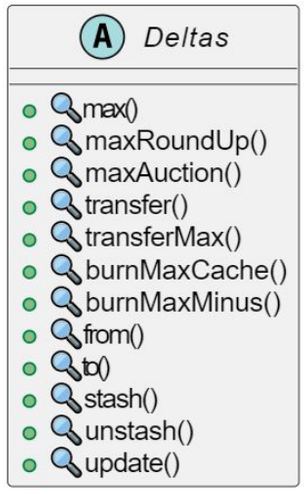


UML - CoverPool









Findings & Resolutions

ID	Title	Category	Severity	Status
CP-1	Outdated Swap Pricing	Logical Error	Critical	Resolved
CL-1	Users Cannot Burn/Claim Due to Underflow	Underflow	Critical	Resolved
DT-1	Errant Deltas.to Calculation	Typo	Critical	Resolved
EP-1	Pool Bricked Due To Cleared liquidityDelta	Logical Error	Critical	Resolved
PS-1	Fully Filled Auction Double Counted	Logical Error	Critical	Resolved
EP-2	Incomplete amountOutDelta Rollover	Logical Error	Critical	Resolved
EP-3	Invalid Epoch Stamping on Pool1	Logical Error	Critical	Resolved
EP-4	Invalid Tick Resulting From TWAP Ratelimiting	Logical Error	Critical	Resolved
CL-2	AmountOutDeltaMax Double Counted In Section2	Logical Error	Critical	Resolved
EP-5	Liquidity Double Counted At Position End	Logical Error	Critical	Resolved
CL-3	Stolen Deltas	Logical Error	Critical	Resolved
CL-4	Locked Liquidity Due To Rounding	Underflow	High	Resolved
PS-2	The SafetyWindow Can Be Circumvented	Logical Error	High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
PS-3	Min Auction Amount Adjusted Twice	Logical Error	High	Resolved
DT-3	Incorrect Output Amount On Overlapping Positions	Logical Error	High	Resolved
EP-6	Uncrossed Ticks Are Set In The EpochMap	Logical Error	High	Resolved
CL-5	Rounding Up In Section5	Logical Error	Medium	Resolved
TK-1	Unused State Variable For Safety Check	Validation	Medium	Resolved
EP-7	Reference Pool Tick Always Rounded Down	Logical Error	Medium	Resolved
GLOBAL-1	Use of block.number on Arbitrum	Compatibility	Medium	Resolved
CP-2	Read-only Reentrancy	Reentrancy	Medium	Resolved
CPF-1	No Minimum TWAP Length	Validation	Medium	Resolved
CP-3	Users Can Update Other Positions	Access Control	Medium	Resolved
CP-4	Unexpected Behavior When Minting	Unexpected Behavior	Low	Resolved
CP-5	Outdated Docs	Documentation	Low	Resolved
ST-1	Use Of Transfer To Send Ether	Best Practices	Low	Pending

Findings & Resolutions

ID	Title	Category	Severity	Status
TK-2	Unnecessary storage manipulation	Optimization	Low	Resolved
CP-6	Unused CollectParams.to	Unused Feature	Low	Resolved
EP-8	Lack of underflow protection	Underflow	Low	Resolved
EP-9	Superfluous Price Assignment	Optimization	Low	Pending
TK-3	Revert Rather Than No-op On priceLimit	Optimization	Low	Resolved
TK-4	Superfluous nextTickPrice Variable	Optimization	Low	Resolved
GLOBAL-2	Unused Q128 Variable	Optimization	Low	Pending
GLOBAL-3	SafeCast	Overflow	Low	Pending
GLOBAL-4	Variables Could Be Made Immutable	Optimization	Low	Resolved
EP-10	syncLatest Simplifications	Optimization	Low	Resolved

CP-1 | Outdated Swap Pricing

Category	Severity	Commit	Location	Status
Logical Error	● Critical	609f5b3024a695aae4bb0ab395555959dfef4ce9	CoverPool.sol: 197	Resolved

Description PoC

The `PoolState memory pool` variable is loaded into memory before the `pool0` or `pool1` storage variables are updated by `syncLatest`. This can yield an outdated `pool.price` when the `syncLatest` call would have updated the relevant pool price.

When this outdated `pool.price` is used to compute the `maxDx` or `maxDy`, it can result in an unchecked underflow allowing virtually any amount of tokens to be swapped in a single auction. Therefore the entirety of the pool's liquidity may be used at the current tick, regardless of the ranges each LPer wanted their position to be active over.

Recommendation

Load the memory pool variable into memory after the `syncLatest` function is called.

Resolution

Poolshark Team: The recommendation was implemented in commit 8414f63.

CL-1 | Users Cannot Burn/Claim Due To Underflow

Category	Severity	Commit	Location	Status
Underflow	● Critical	609f5b3024a695aae4bb0ab395555959dfef4ce9	Claims.sol: 162	Resolved

Description PoC

When users attempt to burn their position at a valid `claimTick`, the transaction reverts due to an underflow as the `cache.finalDeltas.amountOutDeltaMax` is greater than the `amountOutDeltaMax` stored on the position's end tick.

When burning the `cache.finalDeltas` from the `updateTick`, the `cache.finalDeltas.amountOutDeltaMax` should never be greater than the `updateTick.deltas.amountOutDeltaMax`.

Recommendation

Ensure that the `cache.finalDeltas.amountOutDeltaMax` is never greater than the `updateTick.deltas.amountOutDeltaMax` when burning the `cache.finalDeltas` from the `updateTick.deltas`.

Additionally, add logic in `Deltas.burn` to protect against underflow in the event that rounding would result in an underflow revert and prevent users from burning.

Resolution

Poolshark Team: The root cause of the underflow was fixed in 79e2bb6 and the `Deltas.burn` underflow protection was implemented in 74e646b.

DT-1 | Errant Deltas.to Calculation

Category	Severity	Commit	Location	Status
Typo	● Critical	609f5b3024a695aae4bb0ab395555959dfef4ce9	Deltas.sol: 146	Resolved

Description PoC

In `Deltas.to` the `fromDeltas.amountOutDeltaMax` is added to the `toTick.deltas.amountOutDelta`. Since an `amountOutDeltaMax` value is treated as an `amountOutDelta` value, more `amountOut` is errantly attributed to the `toTick`.

This can prevent users from burning as more funds may be attempted to be transferred than are in the contract. Furthermore, in many cases, assets that belong to other users' positions will be transferred out, potentially causing catastrophic loss.

Recommendation

Replace the `toTick.amountOutDelta += fromDeltas.amountOutDeltaMax` with `toTick.deltas.amountOutDelta += fromDeltas.amountOutDelta`.

Resolution

Poolshark Team: The recommendation was implemented in commit b924d61.

EP-1 | Pool Bricked Due To Cleared liquidityDelta

Category	Severity	Commit	Location	Status
Logical Error	● Critical	609f5b3024a695aae4bb0ab395555959dfef4ce9	Epochs.sol: 419	Resolved

Description PoC

When the price from the TWAP reverses directions, the active liquidity is not “stashed” on the `stopTick`.

This can lead to the `CoverPool` being bricked in the following scenario:

- The TWAP price begins at tick 20.
- Alice creates a position from tick 0 to tick -60 in `pool0`.
- The TWAP price goes down to tick -20, entering Alice’s position.
- Because tick 0 is a cross tick while syncing, the `liquidityDelta` on it will be cleared.
- Now the TWAP price goes back up to tick 0, and the liquidity for `pool0` is zeroed out without being stashed.
- As the TWAP price goes down and crosses tick -60, in the `_cross` function the `currentLiquidity` will be 0 but the `liquidityDelta` on the lower tick of her position will still exist. As a result, the end tick `liquidityDelta` subtracted from zero will underflow for `uint`.

The end result is that the `CoverPool` is bricked once the TWAP tick goes below Alice’s lower tick.

Recommendation

When reversing directions and zeroing out the `pool.liquidity`, “stash” this remaining liquidity onto the `stopTick` so that it may be reactivated when the TWAP price continues in that direction.

Resolution

Poolshark Team: The recommended fix was implemented in 79e2bb6.

PS-1 | Fully Filled Auction Double Counted

Category	Severity	Commit	Location	Status
Double Counting	● Critical	b924d617e2e72d354e6dab9df41ba4cd39355826	Positions.sol: 298	Resolved

Description PoC1 PoC2

In cases where an auction is fully filled at the time of claiming, the position ought to be shrunk so that the user is not able to claim again for this auction as a past auction.

However, since the position continues to include the previously filled auction tick, users who claim from a currently filled auction are credited with more tokens then they should be, effectively stealing from others in the pool.

Recommendation

When a user is claiming from a fully filled auction, shrink the position so that the user is not errantly credited with more tokens than they should be.

Resolution

Poolshark Team: The recommendation was implemented in commit f4c6cf1.

EP-2 | Incomplete amountOutDelta Rollover

Category	Severity	Commit	Location	Status
Logical Error	● Critical	9f7bc095b67d320604ff238b234da28f24b7fdcf	Epochs.sol: 341, 366	Resolved

Description PoC

When syncs jump multiple ticks at a time and the direct `nextTickToAccum` does not exist in the `TickMap`, it is possible for users to experience significant loss of assets when the `amountOutDelta` calculations in `_rollover` are restricted to the range between the `pool.price` and the `crossPrice`.

This is because the range restriction in `_rollover` leaves out potentially several ticks that should be accounted for in the `amountDelta` calculations.

Recommendation

Consider initializing the `nextTickToAccum0` and `nextTickToAccum1` ticks if they don't exist in the `TickMap` when creating the cache in `syncLatest`.

Alternatively, when TWAP updates span more than the `tickSpread`, include an `amountOutDelta` calculation from the auction starting price to the `accumPrice` in `_rollover`.

Resolution

Poolshark Team: The recommendation was implemented in commit b87f161.

EP-3 | Invalid Epoch Stamping on Pool 1

Category	Severity	Commit	Location	Status
Typo	● Critical	b19e217d49fefefdbe714b0654011ae1837864d5	Epochs.sol: 156-158	Resolved

Description

During `syncLatest`, `pool1` is utilizing `pool0` ticks when updating the `EpochMap`. In many cases this results in users being locked into their positions and unable to exit due to their end tick being unclaimable and a previous `claimTick` yielding an underflow.

```
if (cache.nextTickToAccum0 > cache.stopTick0
                 && ticks0[cache.nextTickToAccum0].liquidityDeltaMinus > 0) {
                EpochMap.set(tickMap, cache.nextTickToAccum0, state.accumEpoch);
}
```

Recommendation

Use `pool1` ticks to update the ticks for `pool1`:

```
if (cache.nextTickToAccum1 < cache.stopTick1
                 && ticks1[cache.nextTickToAccum1].liquidityDeltaMinus > 0) {
                EpochMap.set(tickMap, cache.nextTickToAccum1, state.accumEpoch);
}
```

Resolution

Poolshark Team: The recommendation was implemented in commit a77bd18.

EP-4 | Invalid Tick Resulting From TWAP Ratelimiting

Category	Severity	Commit	Location	Status
Logical Error	● Critical	609f5b3024a695aae4bb0ab395555959dfef4ce9	Epochs.sol: 259	Resolved

Description PoC

When the `state.lastBlock - state.auctionStart` is not an exact multiple of the `auctionLength`, the resulting `maxLatestTickMove` is not a multiple of the `tickSpread`.

This results in positions being created on invalid ticks and swaps receiving more than the available liquidity for the active auction.

Recommendation

Adjust the `maxLatestTickMove` so that it is a valid multiple of the `tickSpread`.

Resolution

Poolshark Team: The recommendation was implemented in commit f0d52ad.

CL-2 | amountOutDeltaMax Double Counted In Section2

Category	Severity	Commit	Location	Status
Double Counting	● Critical	b924d617e2e72d354e6dab9df41ba4cd39355826	Claims.sol: 263	Resolved

Description PoC

When instantiating the cache using `Claims.getDeltas` in `Positions.update` the `amountOutDeltaMaxStashed` is unstashed onto the `cache.deltas.amountOutDeltaMax`.

The `cache.deltas.amountOutDeltaMax` is ultimately removed from the position's end tick.

However, in `section2` this same value is removed from the position's end tick a second time, therefore perturbing the position's accounting and locking the position in the pool.

Recommendation

Do not remove the same `amountOutDeltaMax` value from the position's end tick in `section2`.

E.g. remove the following line from `section2`:

```
params.zeroForOne ? ticks[params.lower].deltas.amountOutDeltaMax -= amountOutUnfilledMax
                  : ticks[params.upper].deltas.amountOutDeltaMax -= amountOutUnfilledMax;
```

Resolution

Poolshark Team: The recommendation was implemented in commit c9a3a42.

EP-5 | Liquidity Double Counted At Position End

Category	Severity	Commit	Location	Status
Double Counting	● Critical	b924d617e2e72d354e6dab9df41ba4cd39355826	Epochs.sol: 130, 213	Resolved

Description PoC

It is possible for the pool to have active liquidity after the end of a position because the `liquidityDeltaMinus` double counts a portion of the current active liquidity stashed on the `stopTick`.

Consider the following scenario in pool0:

1) Alice creates a position from tick -20 to tick -60.
2) The TWAP goes down to tick -40, entering Alice's position.
3) The TWAP goes up to tick -20. During the `stash`, the pool's active liquidity is added to the `liquidityDelta` on the `stopTick`, -60. Additionally, Alice's `liquidityDeltaMinus` is added to the `stopTick`, double counting her active liquidity.
4) The TWAP goes down to tick -60. Although this is the end of Alice's position, `pool0.liquidity` is now non-zero as her liquidity was stashed onto this tick twice, negating the negative `liquidityDelta` at the end of her position.

Recommendation

When crossing the end of a position, account for the extra `liquidityDelta` by subtracting `stashTick.liquidityDeltaMinus`. Otherwise, remove the `liquidityDeltaMinus` altogether.

Resolution

Poolshark Team: The recommendation was implemented in commit b8d5c44.

CL-3 | Stolen Deltas

Category	Severity	Commit	Location	Status
Logical Error	● High	591e93763698b81337567fe3b235df3aa12b2968	Claims.sol: 144	Resolved

Description PoC

A user is able to steal a portion of another user's tokens due to their `amountOutDeltaMax` being considered during a claim even if they did not contribute to the current `amountOutDelta` on the `claimTick`. As a result, the `percentOutDelta` calculation will attribute tokens for a user when they should not be.

Consider the following scenario:

1) Price is at tick 0
2) Bob mints a position for 100 tokens from 20 to 60
3) Price goes past Bob's claim tick and then back down to tick 20
4) Alice mints a position for 100 tokens from 40 to 60
5) `amountOutDeltaMax` on tick 60 is 200 and `amountOutDelta` is 100
6) Bob burns his entire liquidity with claim tick 60 but only receives 50% of his tokens
7) Alice burns her entire liquidity afterwards and receives 150 tokens, stealing 50 tokens from Bob.

Recommendation

Create another parameter to ignore some `amountOutDeltaMax` if a user has not contributed to a tick's `amountOutDelta`, similar to the paradigm between `liquidityDelta` and `liquidityDeltaMinus`.

Resolution

Poolshark Team: The recommendation was implemented in commit 72f87d3.

CL-4 | Locked Liquidity Due To Rounding

Category	Severity	Commit	Location	Status
Underflow	● High	591e93763698b81337567fe3b235df3aa12b2968	Claims.sol: 281	Resolved

Description PoC

Due to rounding in section3, in some cases users will not be able to burn all of their liquidity since amountOutRemoved is 1 wei greater than the amountOutDeltaMax stored on the position's end tick. This will lead to the user's tx reverting with underflow if the user burns for most of their liquidity.

Recommendation

Consider performing an explicit check to see whether amountOutRemoved is greater than the tick's amountOutDeltaMax. If so, set the amountOutDeltaMax to 0.

Resolution

Poolshark Team: The recommendation was implemented in commit ba281fa.

PS-2 | The SafetyWindow Can Be Circumvented

Category	Severity	Commit	Location	Status
Logical Error	● High	7cc13f9aa74062d56a1d9fef9436c386603ea55e	Positions.sol: 69-73	Resolved

Description

Users may still create positions that begin inside of the safetyWindow due to the following check:

```
if (params.zeroForOne) {
    if (params.lower > cache.requiredStart) revert PositionInsideSafetyWindow();
} else {
    if (params.upper < cache.requiredStart) revert PositionInsideSafetyWindow();
}
```

The validation occurs on the position's end tick rather than the start tick. Therefore users can create positions that begin before the cache.requiredStart. In the event that the position's start tick is at or before the state.latestTick, it will be adjusted to the state.latestTick +/- state.tickSpread. This adjustment will still lie within the safetyWindow. However, if the safetyWindow validation above is corrected, this adjustment logic can be removed as users should never be able to create positions where the beginning of the position is before or equal to the state.latestTick.

Recommendation

Correct the above validation to compare against the params.upper in the pool0 case and the params.lower in the pool1 case.

Alternatively correct the shrinking logic to appropriately shrink the beginning of the position to outside of the safetyWindow by adjusting it to the requiredStart if it is before the requiredStart.

Resolution

Poolshark Team: The recommendation was implemented in commit a02cb7c and 113a0e0.

PS-3 | Min Auction Amount Adjusted Twice

Category	Severity	Commit	Location	Status
Logical Error	● High	fc2c3174d04592cccff172d1472487a8b4685924	Positions.sol: 146	Resolved

Description

In the `Positions.validate` function, `minAmountPerAuction` has already been adjusted to `token1` precision when line 146 is reached, yet `minAmountPerAuction` is adjusted once again to `token1` precision. As a result, `minAmountPerAuction` will be significantly smaller than intended (often 0), and the validation will be rendered useless.

Consider the following scenario:

- `minAmountPerAuction` = 1e18
- `token1Decimals` = 6
- `minAmountPerAuction` = 1e18 / 1e12 / 1e12 = 0

Recommendation

Remove the second adjustment.

Resolution

Poolshark Team: The recommendation was implemented in commit 113a0e0.

DT-3 | Incorrect Output Amount On Overlapping Positions

Category	Severity	Commit	Location	Status
Logical Error	● High	6ef19df4996548e15d0baa656268f97b19454554	Deltas.sol: 95, 97	Resolved

Description PoC

It is possible for a user to be credited with filled amounts that do not belong to them from a previous auction. This is because a stashed tick contains the `deltaMax` values for all positions — regardless of if they had already claimed from the stashed auction result.

Consider the following scenario:

1) Alice mints a position from tick 20 to tick 80 for 100 tokens.
2) Bob mints a position from tick 20 to tick 60 for 100 tokens.
3) The TWAP moves up to tick 20, 83 tokens are swapped.
4) The TWAP moves to tick 40, Bob burns half his liquidity receiving 50 `tokenIn` and 25 `tokenOut`.
5) The TWAP moves to tick 60, Bob burns his remaining liquidity. Bob receives 9 more `tokenIn` although his entire share of the auction was already claimed.
6) The TWAP moves to tick 80, Alice closes her position. Alice receives 24 `tokenIn` although her share should have been 33 `tokenIn`. Bob took 9 of Alice's `tokenIn`.

Recommendation

Account for users having already claimed from past auctions or creating positions and being errantly credited with filled amounts from past auctions. Otherwise document this behavior and implement a "safety window" so that this mechanism cannot be harnessed to vamp filled amounts from LPers.

Resolution

Poolshark Team: The suggested "safety window" was implemented in commit 36cb7a4.

EP-6 | Uncrossed Ticks Are Set In The EpochMap

Category	Severity	Commit	Location	Status
Logical Error	● High	609f5b3024a695aae4bb0ab395555959dfef4ce9	Epochs.sol: 74	Resolved

Description PoC

The `cache.nextTickToAccum0` is set in the `EpochMap` even when the `cache.nextTickToAccum0` is not crossed e.g. it is past the `stopTick0`.

Therefore users are unable to claim at the right tick and are able to claim at a tick that has not yet been accumulated to, perturbing the pool accounting.

Recommendation

Only set the `cache.nextTickToAccum0` in the `EpochMap` if it is being crossed into.

Resolution

Poolshark Team: The recommendation was implemented in commit 74e646b.

CL-5 | Rounding Up In Section5

Category	Severity	Commit	Location	Status
Logical Error	Medium	ba281fa8eff7e36d43fe0f89919d06fcc5fc03fb	Claims.sol: 396	Resolved

Description PoC

In `section5`, an extra wei may be added to the position's `amountOut` due to rounding up. Therefore, in some cases more funds are attempted to be transferred than are in the contract. Otherwise, 1 wei is taken from another user's position.

Recommendation

Consider switching to `Deltas.max` or perform explicit handling.

Resolution

Poolshark Team: The recommendation was implemented in commit e308d31.

TK-1 | Unused State Variable For Safety Check

Category	Severity	Commit	Location	Status
Validation	Medium	b924d617e2e72d354e6dab9df41ba4cd39355826	Ticks.sol: 149	Resolved

Description

`state.liquidityGlobal` is never set, so the liquidity overflow validation only catches amounts greater than `type(int128).max`.

Recommendation

Set the `state.liquidityGlobal`.

Resolution

Poolshark Team: `state.liquidityGlobal` is now updated in commit d15bb47.

EP-7 | Reference Pool Tick Always Rounded Down

Category	Severity	Commit	Location	Status
Logical Error	Medium	609f5b3024a695aae4bb0ab395555959dfef4ce9	Epochs.sol: 249	Resolved

Description

The TWAP tick of the reference pool is always rounded down. It may be beneficial to round up in certain cases to achieve a more accurate price point, and increase the speed of liquidity unlocking.

Recommendation

Consider rounding the TWAP tick of the reference pool to the nearest valid tick rather than always rounding down to the lower valid tick.

Resolution

Poolshark Team: The TWAP tick now shifts by quartiles in commit db9e57e.

GLOBAL-1 | Use Of block.number On Arbitrum

Category	Severity	Commit	Location	Status
Compatibility	Medium	0c9af263973d874c4decaeecb0ad05297cf0414d	Global	Resolved

Description

Throughout the codebase, `block.number` is used to perform syncs and determine `auctionDepth`.

However, `block.number` is synced with the mainnet block number every minute. Therefore less syncs will occur and the `auctionDepth` can be out of date on the order of ~4 blocks at the maximum.

Recommendation

Consider using `ArbSys(100).arbBlockNumber()` to rely on Arbitrum block numbers that are available in real-time.

Resolution

Poolshark Team: `block.timestamp` was adopted to replace `block.number` in commit 116830b.

CP-2 | Read-Only Reentrancy

Category	Severity	Commit	Location	Status
Reentrancy	Medium	6ef19df4996548e15d0baa656268f97b19454554	CoverPool.sol	Resolved

Description

In the `mint`, `burn` and `swap` functions, the `globalState` storage variable is only updated after a token has been transferred to the recipient.

If the token is an ERC777 token and the receiver implements the `tokensReceived` hook, a potential read-only reentrancy arises in the `quote` function because protocol parameters such as `latestPrice`, `auctionStart`, and others are out of date.

Recommendation

Utilize the Check-Effects-Interactions pattern

Resolution

Poolshark Team: The recommendation was implemented in commit 8b7eda9.

CPF-1 | No Minimum TWAP Length

Category	Severity	Commit	Location	Status
Validation	Medium	0c9af263973d874c4decaeecb0ad05297cf0414d	CoverPoolFactory.sol: 30	Resolved

Description

Currently, there is no minimum bound on the `twapLength`, however, an insufficiently large `twapLength` will lead to viable oracle manipulation attacks.

Additionally, a `twapLength` of 0 will break the protocol, causing a panic revert when computing the `averageTick` in `_calculateAverageTick`.

Recommendation

Implement a minimum `twapLength`.

Resolution

Poolshark Team: The recommendation was implemented in commit 2e9d570.

CP-3 | Users Can Update Other Positions

Category	Severity	Commit	Location	Status
Validation	Medium	873b841fbe4a65324384bd584a020f5eefc0cfaa	CoverPool.sol: 111	Resolved

Description

Users are able to update arbitrary positions as the `UpdateParams` in the add function utilize `params.to` rather than the `msg.sender` as the owner.

This way users can manipulate the fill percentage of others by forcing them to update their position when the pool fill percentage is unfavorable.

Recommendation

Use the `msg.sender` as the owner for the `Positions.update` call.

Resolution

Poolshark Team: The recommendation was implemented in commit c703e95.

CP-3 | Unexpected Behavior When Minting

Category	Severity	Commit	Location	Status
Unexpected Behavior	● Low	0c9af263973d874c4decaeecb0ad05297cf0414d	CoverPool.sol: 114	Resolved

Description

When a user mints to add to their existing position that was already crossed into, they end up with two positions: the previous position that was shrunk upon the `Positions.update`, and a newly minted positions that spans the original range.

This may lead to confusion as users may have expected to end up with a single position with added liquidity, rather than two separate positions.

Recommendation

Consider if the `mint` function should add liquidity to the newly shrunk position, otherwise ensure the existing behavior is well documented.

Resolution

Poolshark Team: This is the expected behavior and it will be well documented.

CP-4 | Outdated Docs

Category	Severity	Commit	Location	Status
Documentation	● Low	0c9af263973d874c4decaeecb0ad05297cf0414d	CoverPool.sol: 255	Resolved

Description

In the documentation for the `swap` function, it is stated that “The router must prefund this contract…”, however the `swap` function transfers in the `amountIn` with a call to `_transferIn`.

Recommendation

Update the docs for the `swap` function.

Resolution

Poolshark Team: The outdated docs were removed in commit 1ef04c0.

ST-1 | Use Of Transfer To Send Ether

Category	Severity	Commit	Location	Status
Best Practices	● Low	0c9af263973d874c4decaeecb0ad05297cf0414d	SafeTransfers.sol: 74	Pending

Description

Although currently only ERC20 tokens are supported, the protocol would be incompatible with other contracts, arbitrageurs, and multisig functions if it sent Ether due to the use of `transfer` in `SafeTransfers._transferOut`.

`transfer` forwards only 2300 gas to protect from reentrancy, however, this hard-coded gas limit should be avoided as other protocols and contracts building on top of Poolshark may consume more than 2300 gas in their `fallback`/`receive` function.

Note that there are some multi-sig wallets that use more than 2300 gas in the `fallback` function.

Additionally, gas prices for certain opcodes may change in the future which would force `fallback`/`receive` functions that currently consume <2300 gas to consume >2300 gas and therefore become incompatible.

In the event that a contract cannot receive Ether due to this gas limitation, it may result in loss of funds.

Recommendation

Consider using `call` with a configurable gas limit that can be set sufficiently high and adding a `lock` modifier everywhere these transfer functions are used and can potentially reenter.

Resolution

Pending fix.

TK-2 | Unnecessary storage manipulation

Category	Severity	Commit	Location	Status
Optimization	● Low	0c9af263973d874c4decaeecb0ad05297cf0414d	Ticks.sol: 267-295	Resolved

Description

In the `Ticks.remove` function, when `removeUpper` and `removeLower` are `false`, there are unnecessary storage manipulations that result in no net changes to the `ticks` mapping.

Recommendation

Only do these storage reads and writes inside of the conditionals where the `tickLower` and `tickUpper` are modified.

Resolution

Poolshark Team: The recommendation was implemented in commit 9de5d31.

CP-5 | Unused CollectParams.to

Category	Severity	Commit	Location	Status
Unused Feature	● Low	113a0e04067ef43222d38c5f440f769189a70321	CoverPool.sol: 307	Resolved

Description

In the `_collect` function the `CollectParams.to` value is not used, and rather the claimed amount is always sent to the `msg.sender`.

Recommendation

Use the `CollectParams.to` address when transferring claimed amounts to the user.

Resolution

Poolshark Team: The recommendation was implemented in commit e18ee82.

EP-8 | Lack Of Underflow Protection

Category	Severity	Commit	Location	Status
Underflow	● Low	97047ff98877d45c1ab8483404f844e8bea83892	Epochs.sol: 360, 367	Resolved

Description

In the `_rollover` function, there is underflow protection for the maxes for `pool0`, but not for `pool1`.

Recommendation

Add underflow protection for `pool1`.

Resolution

Poolshark Team: The underflow protection for `pool1` was implemented in commit 6d12f2d.

EP-9 | Superfluous Price Assignment

Category	Severity	Commit	Location	Status
Optimization	Low	97047ff98877d45c1ab8483404f844e8bea83892	Epochs.sol: 301, 316	Pending

Description

In the `_rollover` function, the line `if (state.latestTick == cache.nextTickToCross && crossPrice == pool.price) currentPrice = crossPrice;` simply sets the `currentPrice` to the `pool.price`, however this is already performed 2 lines above.

Recommendation

Remove the superfluous line.

Resolution

Pending Fix

TK-3 | Revert Rather Than No-op On priceLimit

Category	Severity	Commit	Location	Status
Optimization	Low	97047ff98877d45c1ab8483404f844e8bea83892	Ticks.sol: 42	Resolved

Description

Rather than returning a default state and allowing execution to continue, the tx should revert in the case that the `priceLimit` is unsatisfiable.

Recommendation

Consider reverting rather than returning default values, or choose to revert in the `swap` function when the `amountOut` is 0.

Resolution

Poolshark Team: This behavior allows users to still trigger a `syncLatest` in the event that their `priceLimit` is unsatisfied.

TK-4 | Superfluous nextTickPrice Variable

Category	Severity	Commit	Location	Status
Optimization	Low	0c9af263973d874c4decaeecb0ad05297cf0414d	Ticks.sol: 42-43	Resolved

Description

The `nextTickPrice` variable is assigned to the `state.latestPrice` and then immediately the `nextPrice` is assigned to the `nextTickPrice`. The `nextTickPrice` is then never referenced again.

Recommendation

Remove the `nextTickPrice` variable and assign the `nextPrice` to the `state.latestPrice` directly.

Resolution

Poolshark Team: The recommendation was implemented in commit 3a0d67a.

GLOBAL-2 | Unused Q128 Variable

Category	Severity	Commit	Location	Status
Optimization	Low	0c9af263973d874c4decaeecb0ad05297cf0414d	Global	Pending

Description

The `Q128` constant is never referenced after assignment in the `Ticks` `Epochs` and `Positions` libraries.

Recommendation

Remove the `Q128` constant.

Resolution

Pending Fix

GLOBAL-3 | SafeCast

Category	Severity	Commit	Location	Status
Overflow	● Low	0c9af263973d874c4decaeecb0ad05297cf0414d	Global	Pending

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

Pending Fix

GLOBAL-4 | Variables Could Be Made Immutable

Category	Severity	Commit	Location	Status
Optimization	Low	0c9af263973d874c4decaeecb0ad05297cf0414d	Global	Resolved

Description

Variables such as `tickSpread` and `auctionLength` in the global state can be made `immutable`.

Recommendation

Declare these variables `immutable`.

Resolution

Poolshark Team: The recommendation was implemented in commit 4346660.

EP-10 | syncLatest Simplifications

Category	Severity	Commit	Location	Status
Optimization	● Low	0c9af263973d874c4decaeecb0ad05297cf0414d	Epochs.sol	Resolved

Description PoC

The `syncLatest` function can be simplified in the following ways:

1) The additional `_cross` function call for the `stopTick` can be deduplicated if the check inside the `while` loop is modified from `cache.nextTickToAccum0 > cache.stopTick0` to `cache.nextTickToAccum0 >= cache.stopTick0`.
2) The two cases where the `stopTick` is set in the `TickMap` if `newLatestTick > state.latestTick` or `newLatestTick < state.latestTick` can be consolidated into one `newLatestTick != state.latestTick` condition.
3) The increment and decrement of `stopTick.liquidityDelta` by `liquidityDeltaMinus` can be removed as there is no net effect.

Recommendation

Implement the suggested simplifications.

Resolution

Poolshark Team: The suggested simplifications were implemented in 8edd1ce.

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