GA GUARDIAN

Baseline

Fixed Supply Updates

Security Assessment

February 27th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, Osman Ozdemir,

Mark Jonathas, Wafflemak, Michael Lett

Client Firm Baseline

Final Report Date February 27, 2025

Audit Summary

Baseline engaged Guardian to review the security of their Fixed supply updates. From the 17th of February to the 21st of February, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 4 High severity issues were uncovered and promptly addressed by the Baseline team.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Base**

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

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/Baseline-Perps

Table of Contents

Project Overview

Project Summary

Project Name	Baseline
Language	Solidity
Codebase	https://github.com/0xBaseline/baseline-v2
Commit(s)	Initial commit: dca62bb65e86aa35af34b51366183ece3dbc0ee3 Final commit: b74ea9464b2fb28448ebc8e5706b2ef7fb5ee24f

Audit Summary

Delivery Date	February 27, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	4	0	0	1	0	3
Medium	10	0	0	2	0	8
Low	20	0	0	6	0	14

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

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

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

The auditing process pays special attention to the following considerations:

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

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Invalid Remaining Reserves Calculation	Logical Error	High	Resolved
H-02	DoS Of Deployment	DoS	High	Acknowledged
H-03	Extend Interest Stolen	Logical Error	High	Resolved
H-04	All Credit Interests Are Deployed As Liquidity	Logical Error	High	Resolved
M-01	Missing Equality Operator	Logical Error	Medium	Acknowledged
M-02	Outdated Anchor Tick Used In canBump	Logical Error	Medium	Resolved
M-03	Unable To Rebalance Above DISCOVERY_LENGTH	Logical Error	Medium	Resolved
M-04	Temporary DoS Of openPosition()	DoS	Medium	Resolved
M-05	Capacity Errantly Increased	Logical Error	Medium	Acknowledged
M-06	External Liquidity Causes Trade Reverts	Logical Error	Medium	Resolved
M-07	Rebalance Prevented Near Floor Tick	DoS	Medium	Resolved
M-08	Swap With Rebalances Errantly Used	Logical Error	Medium	Resolved
M-09	Donated Liquidity Not Given To Fee Recipient	Logical Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-10	Failed Liquidity Deployment Due To Insufficient Balance	DoS	Medium	Resolved
L-01	Unused Code	Optimization	Low	Resolved
L-02	Lack Of Reentrancy Validation	Reentrancy	Low	Acknowledged
L-03	Swaps Allowed To Non-Baseline Pools	Validation	Low	Resolved
L-04	Superfluous balanceOf Call	Optimization	Low	Resolved
L-05	Payer Param No Longer Used	Optimization	Low	Resolved
L-06	ANCHOR Range Disappears	Validation	Low	Resolved
L-07	onlyKernel Modifier Discrepancy	Validation	Low	Resolved
L-08	Misleading Burn Function	Documentation	Low	Acknowledged
L-09	_canBump Early Return	Optimization	Low	Resolved
L-10	Duplicated BLV Price Getters	Optimization	Low	Acknowledged
L-11	Deadline Set To Block.timestamp	Logical Error	Low	Acknowledged
L-12	Misleading Documentation In getBaselineValue	Documentation	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-13	Unusable DISCOVERY_LENGTH	Superfluous Code	Low	Resolved
L-14	_canBump Validation Does Not Round Correctly	Rounding	Low	Resolved
L-15	Max Tick Discovery Liquidity Warning	Warning	Low	Acknowledged
L-16	Missing payable On exactInputSingleVanilla	Modifiers	Low	Resolved
L-17	Unnecessary Allowance	Optimization	Low	Resolved
L-18	Leverage Can Be Below 1x	Warning	Low	Resolved
L-19	Superfluous Comment	Superfluous Code	Low	Resolved
L-20	Inability To Update BToken Controller	Warning	Low	Acknowledged

H-01 | Invalid Remaining Reserves Calculation

Category	Severity	Location	Status
Logical Error	High	MarketMaking.sol: 357	Resolved

Description

The `MarketMaking` contract verifies if the protocol can bump by simulating an increase in the `blvTick`, and later validating some conditions, like `bumpedCapacity > circulating`.

The `bumpedAnchorCapacity` is calculated based on the `_getAnchorReserves`. However, the `bumpedFloorCapacity` is mistakenly uses the `remainingReserves` as follows:

`int256 remainingReserves = _getVirtualReserves() + reserve.balanceOf(address(BPOOL));`

However, the balance of the `BPOOL` contains all reserves, as they were all removed from the ranges, so `remainingReserves` is actually equal to `totalReserves`.

Recommendation

Subtract the reserves used to calculate the ANCHOR range to correctly determine how many reserves remain.

Resolution

Baseline Team: The issue was fixed in line MarketMaking.sol#L354.

H-02 | DoS Of Deployment

Category	Severity	Location	Status
DoS	High	Deployment	Acknowledged

Description PoC

Deploying with `vm.startBroadcast()` will lead to each call happening as a separate transaction. This gives a user who was distributed `bTokens` the opportunity to interact with the Uniswap pool prior to the first rebalance occurring.

A malicious user can deploy liquidity below the desired `BLV`. Then, they can swap to their new deployed liquidity range.

This will mean the active tick is below the `BLV` tick. When `rebalance()` is invoked, it will attempt to set the lower tick of the anchor range to the `BLV` tick and the upper tick will be calculated using the current active tick.

Since the current active tick is below the `BLV` tick, `setTicks()` will revert due to `InvalidTickRange`. This will DoS the deployment after the tokens are distributed and the contract has been deployed.

Recommendation

Deploy inside a smart contract function, so that the deployment happens atomically.

Resolution

Baseline Team: Acknowledged.

H-03 | Extend Interest Stolen

Category	Severity	Location	Status
Logical Error	High	CreditFacility.sol: 438	Resolved

Description

The Interest accrued from the `extend` function sits in the `CreditFacility` contract until the fee recipient removes it with their approval.

However this poses an issue because the `_swapExactOut` function transfers the entire contract balance of the `CreditFacility` to the `BPOOL` contract.

As a result these fee amounts which are sitting in the `CreditFacility` contract will be deployed into the protocol liquidity instead of collectable.

Recommendation

In the `extend` function, instead of transferring the reserve amount from the user to the `CreditFacility` contract, transfer the reserve amount to the fee recipient directly. Furthermore, be sure there are no other instances where reserves are left in the `CreditFacility` contract.

Resolution

Baseline Team: The issue was fixed in line CreditFacility.sol#L437.

H-04 | All Credit Interests Are Deployed As Liquidity

Category	Severity	Location	Status
Logical Error	High	CreditFacility.sol: 560	Resolved

Description

The `CreditFacility` `_sendReserves` function used to keep the interest reserves amount in the `CreditFacility` contract so that this amount could be removed by the fee receiver which is approved for the `CreditFacility`.

However now, because the `removeAllFrom` function leaves all removed tokens in the BPOOL contract, the interest amount is not collected by the protocol and will instead be deployed back into the liquidity structure.

Recommendation

Use `BPOOL.transferToken(reserve, feeRecipient, _interest);` in the `_sendReserves` function. Additionally, remove the `feeRecipient` approval logic as it is no longer necessary.

Resolution

Baseline Team: The issue was fixed in line CreditFacility.sol#L548.

M-01 | Missing Equality Operator

Category	Severity	Location	Status
Logical Error	Medium	Brouter.sol: 299	Acknowledged

Description

The `_tradingInFloor` functions in Policies verify if the active tick is at or below the floor's upper tick. However, in `Brouter`, this function only contains `<`, so the check will succeed when `activeTick == tickU`.

Recommendation

Update the operator to `<=` so the call reverts when active tick is exactly at the `BLV`.

Resolution

Baseline Team: Acknowledged.

M-02 | Outdated Anchor Tick Used In canBump

Category	Severity	Location	Status
Logical Error	Medium	MarketMaking.sol: 340	Resolved

Description

The `canBump` function uses an outdated `anchorTick` which has not been updated to reflect the current price which the protocol is rebalancing for.

As a result the capacity calculations for the Anchor range are not accurate to what the capacity will actually be after the rebalance.

This will often result in bumping when bumps should not occur, which will often prevent a rebalance from occurring since the final capacity invariant cannot be held. Or, more rarely, not allowing bumps to occur when they ought to be.

Recommendation

Consider updating the `anchorTick` to the latest that will be used in the rebalance.

Resolution

Baseline Team: Resolved.

M-03 | Unable To Rebalance Above DISCOVERY_LENGTH

Category	Severity	Location	Status
Logical Error	Medium	MarketMaking.sol: 202	Resolved

Description

The MarketMaking policy is mainly in charged of rebalancing the liquidity positions, when canRebalance is true.

Normally, a rebalance will be triggered when price moved outside of the rebalance ticks and its within a certain range:

```
bool isWithinRange = activeTick > blvTick & activeTick < anchorTick + DISCOVERY_LENGTH;
```

However, the DISCOVERY liquidity will range from the anchorTick to the MAX_TICK. This suggests that rebalance is not possible when price is inside the DISCOVERY but above anchorTick + DISCOVERY_LENGTH.

Recommendation

Update the withinRange to include all the DISCOVERY range so rebalance is possible when price is at that range:

```
bool isWithinRange = activeTick > blvTick & activeTick < MAX_TICK;
```

Resolution

Baseline Team: The issue was fixed in line MarketMaking.sol#L201.

M-04 | Temporary DoS Of openPosition()

Category	Severity	Location	Status
DoS	Medium	Brouter: 233	Resolved

Description

The `isEth` validation checks the balance of the contract instead of the `msg.value`. A malicious user can send ether to the `Brouter`, in order to trigger a refund to `LoopFacility` when `openPosition()` is called.

Since `LoopFacility` does not have a `receive()` function, this will cause a call to `openPosition()` to revert. The malicious user can then pull their ether out through a swap in the following transaction.

Recommendation

Change the validation for `isEth` from checking the contract balance to checking the `msg.value`.

Resolution

Baseline Team: The issue was fixed in line Brouter.sol#L242.

M-05 | Capacity Errantly Increased

Category	Severity	Location	Status
Logical Error	Medium	CreditFacility.sol: 356	Acknowledged

Description

The `CreditFacility` allows users to borrow reserves with `bAssets` as collateral. These reserves will be retrieved from the Liquidity Ranges.

If there are not enough reserves in the `FLOOR`, the excess amount will be taken from the `ANCHOR`. However, when users repay reserves, these will be added to the `FLOOR` only.

Due to the fact that there is no interest charged, users can deliberately borrow enough reserves to remove some from the `ANCHOR` range and immediately repay.

This will increase the capacity of the system as the reserves that were in the `ANCHOR` had lower capacity than if they are valued at the `blv`. Additionally, this may open arbitrage opportunities, as well as DoS user actions, as liquidity in the trading range is moved down.

Recommendation

Set interest greater than zero, to disincentivize whales from manipulating the reserves and capacity. Additionally, consider triggering a rebalance, if possible, to re distribute the reserves where they belong.

Resolution

Baseline Team: Acknowledged.

M-06 | External Liquidity Causes Trade Reverts

Category	Severity	Location	Status
Logical Error	Medium	Brouter: 266	Resolved

Description

When trading in the floor, liquidity is removed from the floor during a buy in order to improve price movement.

However, an external user can provide liquidity in the floor range, making the trade still revert. This revert will prevent swaps that are bringing the price closer to the BLV from executing.

Recommendation

Instead, validate that the price has moved close to the BLV and check that the price is not in the floor for `closePosition()`.

Resolution

Baseline Team: The issue was fixed in line Brouter.sol#L272.

M-07 | Rebalance Prevented Near Floor Tick

Category	Severity	Location	Status
DoS	Medium	MarketMaking.sol	Resolved

Description

In the `_getACU` function when the active pool price is just above the `blvTick` there will be an overflow when attempting to cast the result of `FullMath.mulDiv(amount1, FixedPoint96.Q96, sqrtRatioBX96 - sqrtRatioAX96)` to a uint128 variable inside of the `getAmount0ForLiquidity` function.

As a result rebalances are DoS'd when the pool price is in this edge case range.

Recommendation

Be aware of this DoS and consider refactoring the leverage calculations to avoid calculating the ACU when the price is close to the `blvTick` and instead returning a default asymptotic value.

Resolution

Baseline Team: Resolved.

M-08 | Swap With Rebalances Errantly Used

Category	Severity	Location	Status
Logical Error	Medium	LoopFacility.sol: 175	Resolved

Description

In the `closePosition` function the `exactInputSingle` function on the `BRouter` contract is used which rebalances before and after the swap.

This does not match the previous behavior of the `closePosition` function and can prevent users from repaying their debts because the rebalance function may revert with an `BackingInsolvent` error.

Recommendation

Use the `exactInputSingleVanilla` function instead.

Resolution

Baseline Team: The issue was fixed in line LoopFacility.sol#L175.

M-09 | Donated Liquidity Not Given To Fee Recipient

Category	Severity	Location	Status
Logical Error	Medium	BPOOL.sol: 223	Resolved

Description

In the `removeAllFrom` the donated liquidity that was not deployed by the protocol is intended to be transferred to the fee recipient.

However in the case where the ranges are updated and third party liquidity is found in the new ranges, the following early return is used in the `removeAllFrom` function:

```
liquidityToRemove = currentLiquidity; if (liquidityToRemove = 0) return (0, bAssetFees_, 0, reserveFees_);
```

In this case the transfers at the end of the `removeAllFrom` function are not made.

```
reserve.safeTransfer(feeRecipient, reserveFees_); bAsset.transfer(feeRecipient, bAssetFees_);
```

Recommendation

Inside the early return case, be sure to make the same transfers to the `feeRecipient`.

Resolution

Baseline Team: The issue was fixed in line BPOOL.v1.sol#L218.

M-10 | Failed Liquidity Deployment Due To Insufficient Balance

Category	Severity	Location	Status
DoS	Medium	MarketMaking.sol	Resolved

Description

During the liquidity deployment phase of rebalancing, `DISCOVERY` liquidity is added first, followed by `ANCHOR` liquidity. The threshold liquidity added to `DISCOVERY` is the minimum of multiple calculations: `uint256(threshold_).min(uint256(bTokenLiquidityMax))`.

When the final threshold liquidity is `bTokenLiquidityMax`, the entire `bAsset` balance of the `BPOOL` will be deployed to the `DISCOVERY` range during `deployLiquidityTo(DISCOVERY)`, as `bTokenLiquidityMax` is calculated using `balanceOf(BPOOL)`.

However, adding liquidity to `ANCHOR` in the next step also requires some `bAssets` when `activeTick < anchorTick`. Since the entire `bAsset` balance has already been deployed to the `DISCOVERY` range, `addReservesTo(ANCHOR)` fails during `uniswapV3MintCallback` due to insufficient balance.

Recommendation

Consider leaving a buffer amount when calculating `bTokenLiquidityMax` instead of using the entire balance. This ensures that the contract retains `bAssets` to add to `ANCHOR`, even when `bTokenLiquidityMax` is deployed to `DISCOVERY`.

Resolution

Baseline Team: The issue was fixed in line MarketMaking.sol#L472.

L-01 | Unused Code

Category	Severity	Location	Status
Optimization	Low	Global	Resolved

Description

The following code is not used in the current implementation:
• `LoopFacility._tradingInFloor()`
• `BlastClaimer`, `IUniswapV3Pool`, `FixedPoint96` import in LOOPS
• debug code (`console2.sol`, `PoolViewerLib.sol`)

Recommendation

Remove the unused code or add an implementation for it.

Resolution

Baseline Team: Resolved.

L-02 | Lack Of Reentrancy Validation

Category	Severity	Location	Status
Reentrancy	Low	Brouter: 102, 113, 126 & 137	Acknowledged

Description

_swap() sends ether to the user via call(), which will hand over the execution flow to the receive() function if it is a smart contract. It is best practice to use reentrancy modifiers when this occurs.

Recommendation

Add reentrancy guard modifiers to the swap functions.

Resolution

Baseline Team: Acknowledged.

L-03 | Swaps Allowed To Non-Baseline Pools

Category	Severity	Location	Status
Validation	Low	Brouter: 102, 113, 126 & 137	Resolved

Description

`_swap()` does not validate the fee tier that is passed for a swap. This allows users to perform swaps with pools that have been created with the same tokens but set to different fee tiers.

Recommendation

Validate the fee of the trade in the `_swap()` function.

Resolution

Baseline Team: The issue was fixed in line Brouter.sol#L234.

L-04 | Superfluous balanceOf Call

Category	Severity	Location	Status
Optimization	Low	MarketMaking.sol: 270	Resolved

Description

The `_removeLiquidity` contains a call to `bAsset.balanceOf(address(BPOOL));` whose return value is not used.

Recommendation

Remove the `balanceOf` call.

Resolution

Baseline Team: Resolved.

L-05 | Payer Param No Longer Used

Category	Severity	Location	Status
Optimization	Low	BPOOL.v1.sol	Resolved

Description

The `BPOOL` contract will always own the reserves and `bAssets` used to add liquidity to ranges, so in the `uniswapV3MintCallback` these tokens are transferred directly to the pool.

Therefore, there is no need to encode the `payer` or `msg.sender` during `pool.mint` as this encoded data is not longer used.

Recommendation

Send empty data in the last param in `pool.mint`

Resolution

Baseline Team: Resolved.

L-06 | ANCHOR Range Disappears

Category	Severity	Location	Status
Validation	Low	MarketMaking.sol: 315	Resolved

Description

The rebalance action can now be executed when more than 8 hours have passed since the last rebalance, bypassing the other price and range checks.

This allows the ANCHOR range to disappear when the following scenarios are met:

• `activeTick` = `blvTick`
• (`activeTick` < `blvTick` + 199) & (`liquidityA` > `_getThresholdLiquidity()`)

Although the first scenario might be expected, the second one might not, as it will create a DISCOVERY position with reserves.

Recommendation

Consider if this is the expected behavior and prevent rebalances to occur.

Resolution

Baseline Team: Resolved.

L-07 | onlyKernel Modifier Discrepancy

Category	Severity	Location	Status
Validation	Low	Global	Resolved

Description

The `onlyKernel` modifier prevents external calls to certain functions when Modules are installed or Policies are activated.

However, this modifier is only used in certain cases, leaving some unprotected functions, like the `configureDependencies` in Policies, that can lead to unexpected scenarios.

Recommendation

Consider adding the `onlyKernel` modifier to the all Module and Policies functions that should only be called by the Kernel contract.

Resolution

Baseline Team: Resolved.

L-08 | Misleading Burn Function

Category	Severity	Location	Status
Documentation	Low	CREDT.v1.sol	Acknowledged

Description

The `CREDT` module still contains a `_burnDefaultedCollateral`, but the bAssets are not burned anymore. Instead, they are transferred to the `BPOOL` module to be used for the next liquidity rebalance. Although the function does not actually burn, it can be misleading, as well as its natspec.

Recommendation

Update the `_burnDefaultedCollateral` function name as well as the comments, and avoid suggesting a `burn`

Resolution

Baseline Team: Acknowledged.

L-09 | _canBump Early Return

Category	Severity	Location	Status
Optimization	Low	MarketMaking.sol: 339	Resolved

Description

The rebalance operation, after removing liquidity, will try to check if the current liquidity structure accepts a `bump`, which relies on certain conditions being met at the same time.

One of these conditions is `tickDelta > BUMPABLE_PREMIUM` which prevents bumps if the tick premium (difference between the `activeTick` and the `blvTick`) is greater than 1500 (default value).

Therefore, to save gas and avoid more calculations, the function should early return with `false` if this condition is not met.

Recommendation

Early return `false` if `tickDelta = BUMPABLE_PREMIUM`

Resolution

Baseline Team: The issue was fixed in line MarketMaking.sol#L335.

L-10 | Duplicated BLV Price Getters

Category	Severity	Location	Status
Optimization	Low	BPOOL.v1.sol: 272	Acknowledged

Description

Both `BPOOL.getBaselineValue`, `LoopFacility.getBaselineValue()` and `MarketMaking.getBLV()` calculate the current baseline value price based on the upper tick of the floor.

Although they all return the same value, this can lead to issues in the future if one is updated but not the others.

Recommendation

Consider having one single source of truth for the `blv` calculation.

Resolution

Baseline Team: Acknowledged.

L-11 | Deadline Set To Block.timestamp

Category	Severity	Location	Status
Logical Error	Low	Global	Acknowledged

Description

The `Brouter` performs swaps in the `UniswapV3Pool`, used by some Policies. The issue arises when using `block.timestamp` as the deadline parameter for these swaps.

A malicious block builder will be able to execute this at any time, when such transaction is useful for manipulating the price.

Recommendation

Add an optional deadline parameter to functions that routes swaps through the `Brouter` and use this instead of `block.timestamp` for all the swaps.

Resolution

Baseline Team: Acknowledged.

L-12 | Misleading Documentation In getBaselineValue

Category	Severity	Location	Status
Documentation	Low	BPOOL.v1.sol: 271	Resolved

Description

The `getBaselineValue` calculates the `BToken` price at upper tick of the `FLOOR` range. However, the documentation mentions `Returns the price at the lower tick of the floor position`, which is misleading.

Recommendation

Update the documentation to: `Returns the price at the upper tick of the floor position`

Resolution

Baseline Team: The issue was fixed in line BPOOL.v1.sol#L271.

L-13 | Unusable DISCOVERY_LENGTH

Category	Severity	Location	Status
Superfluous Code	Low	MarketMaking.sol	Resolved

Description

The `DISCOVERY_LENGTH` value is assignable by the owner address and is used in `canBump` but actually has nothing to do with the length of the discovery range as it is hardcoded to the max tick.

Recommendation

Consider either removing the `DISCOVERY_LENGTH` variable or making the Discovery range configurable by it.

Resolution

Baseline Team: Resolved.

L-14 | _canBump Validation Does Not Round Correctly

Category	Severity	Location	Status
Rounding	Low	MarketMaking.sol: 350	Resolved

Description

In the `_canBump` function there is validation to check if the circulating supply can be absorbed by the total reserves immediately after increasing the `blvTick`.

```
if (totalReserves < circulating.mulWad(getBLV())) {blvTick = T_S; return false;}
```

This validation multiplies the circulating supply by the BLV price using mulWad, which rounds down. This does not round in the protocol's favor as it is rounding down the circulating value that the reserves must cover.

This is in contrast to the same validation which is performed differently in the `_removeLiquidity` function.

```
     uint256 maxCapacity = totalReserves.divWad(getBLV());

if (maxCapacity < circulating) {revert BackingInsolvent();}
```

In the `_removeLiquidity` function the `maxCapacity` is exposed to rounding down because it is the `totalReserves` divided by the BLV price with `divWad` which rounds down.

This is the correct way to perform this validation which rounds in favor of being more conservative about the capacity invariant.

Recommendation

Use the same validation as is performed in the `_removeLiquidity` which rounds conservatively.

Resolution

Baseline Team: Resolved.

L-15 | Max Tick Discovery Liquidity Warning

Category	Severity	Location	Status
Warning	Low	MarketMaking.sol	Acknowledged

Description

Since the Discovery liquidity is now deployed to the max tick and the circulating supply is now limited, the liquidity achievable in the Discovery range will be somewhat limited.

In many cases this is will result in the capping of the threshold liquidity, which may give resistance to deploying the liquidity structure that is desired.

Recommendation

Be aware of this constraint and be prepared to adjust the discovery range as needed if outcomes are not as desired.

Resolution

Baseline Team: Acknowledged.

L-16 | Missing payable On exactInputSingleVanilla

Category	Severity	Location	Status
Modifiers	Low	Brouter.sol: 126	Resolved

Description

The `exactInputSingleVanilla` function does not have the `payable` keyword, unlike the other three functions, and therefore cannot perform swaps with the native token.

Recommendation

Add `payable` to this function as well.

Resolution

Baseline Team: The issue was fixed in line Brouter.sol#L126.

L-17 | Unnecessary Allowance

Category	Severity	Location	Status
Optimization	Low	MarketMaking.sol: 142	Resolved

Description

The `MarketMaking` contract grants approval to `BPOOL` to spend reserve tokens. However, this approval is unnecessary, as `BPOOL` no longer invokes `transferFrom` after the updates.

Recommendation

Remove floating allowances.

Resolution

Baseline Team: The issue was fixed in line MarketMaking.sol#L115.

L-18 | Leverage Can Be Below 1x

Category	Severity	Location	Status
Warning	Low	MarketMaking.sol: 518	Resolved

Description

In the `_getLeverage` function the leverage is multiplied by 0.9999 to avoid any rounding up edge cases. However this allows the leverage to be lower than 1x, which may be unexpected.

Recommendation

Consider enforcing a minimum value of 1e18 for the leverage result.

Resolution

Baseline Team: The issue was fixed in line MarketMaking.sol#L509.

L-19 | Superfluous Comment

Category	Severity	Location	Status
Superfluous Code	Low	MarketMaking.sol: 410-411	Resolved

Description

The comment related to spot supply invariant check at lines 410–411 of the `MarketMaking` contract still persists, even though the line `uint256 spotSupply = getCirculatingSupply() - LOOPS.totalCollateral() - CREDT.totalCollateralized()` has been removed.

Recommendation

Remove the unnecessary comment.

Resolution

Baseline Team: The issue was fixed in line MarketMaking.sol#L407.

L-20 | Inability To Update BToken Controller

Category	Severity	Location	Status
Warning	Low	BPOOL.v1.sol: 253	Acknowledged

Description

The `BToken` controller is initialized with the `BPOOL` address during deployment, which is also set as the controller address.

The `BToken.setController` has an access control validation so only the current controller address (BPOOL) can update it.

However, the only function in `BPOOL` that can update the controller is the `migrateBToken`, which is `permissioned` and there is no Policy that has the function permission.

Recommendation

Consider adding `migrateBToken` permission to the policy that should be allowed to call this function.

Resolution

Baseline 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