GA GUARDIAN

Baseline Markets

V2 Updates

Security Assessment

June 17th, 2024

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Wafflemakr, Giraffe, Osman Ozdemir

Client Firm Baseline

Final Report Date June 17, 2024

Audit Summary

Baseline engaged Guardian to review the security of its concentrated liquidity protocol, supporting a baseline value for its YES token. From the 27th of May to the 6th of June, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

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

🔗 Blockchain network: **Blast**

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

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

Table of Contents

Project Information

Project Overview 4

Audit Scope & Methodology 5

Smart Contract Risk Assessment

Findings & Resolutions 7

Addendum

Disclaimer 38

About Guardian Audits 39

Project Overview

Project Summary

Project Name	Baseline Markets
Language	Solidity
Codebase	https://github.com/0xBaseline/baseline-v2
Commit(s)	Initial: a37d09ec996318db14cb48e05b916c611b1459f2 Final: 102c7a3abcd34b3c1f97efd400e3a7af4afbaf6b

Audit Summary

Delivery Date	June 17th, 2024
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	12	0	0	4	0	8
Medium	6	0	0	4	0	2
Low	10	0	0	0	0	10

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	DoS Via External Liquidity By Predicting Positions	DoS	High	Resolved
H-02	Slide Allows Anchor To Become Larger Than Discovery	Logical Error	High	Resolved
H-03	getLiquidityForReserve DoS	DoS	High	Resolved
H-04	Leverage And Deleverage Are Sandwhichable	Sandwhich Attack	High	Acknowledged
H-05	removeAllFrom DoS With External Liquidity	DoS	High	Resolved
H-06	Migration Cannot Be Done	DoS	High	Resolved
H-07	Premium Compounding Is Incorrect	Logical Error	High	Resolved
H-08	bAsset Shorts Can Make A Guaranteed Profit	Gaming	High	Acknowledged
H-09	Deleverage Can Break Through Floor Tick	Logical Error	High	Resolved
H-10	Borrows In The Floor Invalidate Baseline Value	Logical Error	High	Resolved
H-11	Shorts Profit Because Of Fixed Anchor Width	Gaming	High	Acknowledged
H-12	Anchor Liquidity Can Be Removed	Logical Error	High	Acknowledged
M-01	Launch Allows For Anchor Larger Than Target Width	Logical Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-02	Floor Liquidity And Reserves Decrease After Sweep	Logical Error	Medium	Acknowledged
M-03	Swap Fees Included In Circulating Supply	Logical Error	Medium	Acknowledged
M-04	Interest Free Credits Intra-Day	Gaming	Medium	Resolved
M-05	Extra Interest Charged When Extending Credit	Logical Error	Medium	Acknowledged
M-06	Decrease Position Prevented	Logical Error	Medium	Acknowledged
L-01	getLiquidityForReserves Function Edge Case Not Handled	Logical Error	Low	Resolved
L-02	Incorrect Comment	Documentation	Low	Resolved
L-03	Superfluous Capacity Calculation	Optimization	Low	Resolved
L-04	Unused LIQUIDITY_THICKNESS Variable	Optimization	Low	Resolved
L-05	Potentially Dangerous liquidityToRemove Assignment	Best Practices	Low	Resolved
L-06	Lack Of safeTransfer	Best Practices	Low	Resolved
L-07	Outdated Comment	Documentation	Low	Resolved
L-08	Unused State Variables	Superfluous Code	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-09	Inaccurate Return Value Name	Best Practices	Low	Resolved
L-10	Change In Credit On Repay Does Not Match Change In Reserves	Documentation	Low	Resolved

H-01 | DoS Via External Liquidity By Predicting Positions

Category	Severity	Location	Status
DoS	High	Global	Resolved

Description PoC

Users adding external liquidity to the protocol's positions should not affect the market making operations, as the liquidity for each range is tracked by the `getLiquidity` mapping in the `BPOOL` contract storage.

The issue relies on the way this mapping is updated after an operation completes. When using `addReservesTo` or `addLiquidityTo` this mapping will be updated with the new real position liquidity, including externally added liquidity.

An attacker can DoS the `sweep` operation by following these steps:

1. Add ext liquidity to the projected ANCHOR range after a bump (lowerTick + 1 TS, upperTick).
2. Execute `bump()`, now the ANCHOR range is the same as the projected one, which we just maliciously added liquidity to.
3. Now the `addLiquidityTo` function call for the ANCHOR range records the malicious liquidity as belonging to the system.
4. `sweep` now reverts with underflow.

Recommendation

Remove any liquidity that is sitting in positions that will be used, but aren't currently, and add this amount to the `bufferedReserves` value.

Resolution

Baseline Team: The issue was resolved in PR#90.

H-02 | Slide Allows Anchor To Exceed Discovery

Category	Severity	Location	Status
Logical Error	High	MarketMaking.sol: 305	Resolved

Description

When removing and adding back liquidity to the anchor position in the slide function, the `anchorReserves` are tracked and added back to the adjusted anchor position.

However the anchor position has moved leftwards after the slide, therefore the liquidity for the anchor position will increase as the reserves are maintained.

As a result, since the new discovery liquidity cannot increase past the old discovery liquidity, the protocol can enter a state where the anchor liquidity is greater than the discovery liquidity.

Recommendation

Consider the following resolution options:

• No longer require the new discoveryL ≤ old discoveryL
• Switch back to maintaining the liquidity for the anchor, rather than the reserves

Resolution

Baseline Team: The issue was resolved in PR#files#diff-3579424c9b33337cc1a4bf4454c2c5a502df76dc22c5110b8767b5f976fee172R312-R314.

H-03 | getLiquidityForReserve DoS

Category	Severity	Location	Status
DoS	High	BPOOL.v1.sol: 318	Resolved

Description

In the getLiquidityForReserves function there is no validation that the lower price and upper price provided to the getLiquidityForAmount1 function are not equal. This is possible either with an anchor range that has no width or when the active price is exactly that of the lower tick price for a range. This behavior results in a DoS of several key functionalities of the system when these edge cases are hit.

Recommendation

Implement the following validation such that the upperPrice can never be less than or equal to the lower price:

```
function getLiquidityForReserves(
     uint160 _sqrtPriceL,
     uint160 _sqrtPriceU,
     uint256 *reserves
) public view returns (uint128 liquidity*) {
     (uint160 sqrtPriceA,,,,,,) = pool.slot0();
     uint160 upperPrice = min(_sqrtPriceU, sqrtPriceA);
     if (upperPrice <= _sqrtPriceL) { return 0; }
     liquidity_ = LiquidityAmounts.getLiquidityForAmount1(
          _sqrtPriceL,
          upperPrice,
          _reserves
     );
}
```

Resolution

Baseline Team: Resolved.

H-04 | Leverage And Deleverage Are Sandwhichable

Category	Severity	Location	Status
Sandwhich Attack	High	CreditFacility.sol: 228, 324	Acknowledged

Description

In the leverage and deleverage functions traders flash borrow and repay using a swap where the `maxAmountIn` is the borrowed amount for borrows and the unlocked collateral amount for repayments.

This allows malicious actors to frontrun these actions and push the price far enough such that these limits are hit. In the case of borrowing, this leaves the borrower with no funds received from the action as the `reservesNeeded` can be push to the maximum of the entire `newPrincipal_`.

Recommendation

Allow the user to configure an `amountInMaximum` themselves, and do not allow this `amountInMaximum` to be greater than the borrowed amount for borrows and the unlocked collateral amount for repayments.

Additionally, consider allowing the user to configure the `sqrtPriceLimitX96` and deadline on the swap for additional MEV protection.

Resolution

Baseline Team:As the original deployment is planned for Blast this is not an immediate issue, but will be addressed before subsequent chains are supported.

H-05 | removeAllFrom DoS With External Liquidity

Category	Severity	Location	Status
DoS	High	BPOOL.sol: 226	Resolved

Description

In the `removeAllFrom` function, the function is short circuited if there is 0 `liquidityToRemove`, however this value is based upon the amount that is deployed in Uniswap.

Therefore a malicious actor may add to this amount in order to stop this early return check from triggering when the `getLiquidity` mapping reports that there is no liquidity deployed there for the system.

In this scenario the function execution continues on to remove all of the attacker's malicious liquidity from the range and then attempts to remove 0 liquidity from the range after it has been depleted. This second attempt to remove liquidity will revert with the `NP` error from Uniswap.

Recommendation

Add a second early return case right before the `_removeLiquidity` invocation:

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

Resolution

Baseline Team: Resolved.

H-06 | Migration Cannot Be Done

Category	Severity	Location	Status
DoS	High	BaselineInit.sol: 155	Resolved

Description

During the migration process, the owner will construct the initial distribution of spot and credits and it will be done with `allocate` function. This can only be called before the pool is launched, and the pool will be deployed later.

Users might have both spot and credit to be distributed. But a user can also have only spot or only credit. However, the check that ensures the allocation is not empty is incorrect, and the `allocate` function will revert if a user has only spot or only credit.

The whole migration process will be disrupted even if just one user has only spot or only credit.

Recommendation

Change this line

`if (_spot[i] == 0 || _collateral[i] == 0) revert InvalidAllocation();`

to this:

`if (_spot[i] == 0 && _collateral[i] == 0) revert InvalidAllocation();`

Resolution

Baseline Team: The issue was resolved in commit b083ad0.

H-07 | Premium Compounding Is Incorrect

Category	Severity	Location	Status
Logical Error	High	MarketMaking.sol: 365-366	Resolved

Description PoC

The protocol deploys liquidity to ranges based on a liquidity premium and this premium supposed to be compounding when the difference between the `activeTick` and the `floorTickL` increases.

However, due to incorrect usage of the `powWad` in the `getCurrentThreshold` function, premium decreases instead of increasing. The reason of this behaviour is the numerator (`1e16`) in the formula is smaller than `1e18` and it causes result to decrease in every power.

As a result of this, all liquidity managements in `sweep` and `slide` will be incorrect, and the maximum liquidity in discovery range will never pass `1.1 * anchor liquidity` during these operations.

Recommendation

Ensure the formula compounds the premium in a positive way while using the `powWad`, while being sure to implement the correct test coverage for the liquidity threshold, as this bug was missed by the existing testing coverage.

Resolution

Baseline Team: Resolved.

H-08 | bAsset Shorts Can Make A Guaranteed Profit

Category	Severity	Location	Status
Gaming	High	Global	Acknowledged

Description PoC

In the Baseline V2 system it is possible for a user to arbitrage the system in order to make guaranteed profit on a short.

This arbitrage can occur with the following actions:

1. Bob borrows X bAssets from Alice
2. Bob buys Y bAssets to move the anchor up after sweeping
3. Bob sells Y bAssets and achieves a lower pool price than before his buy because:
1. The anchor has moved up, meaning there is less capacity for the same amount of liquidity (though in many cases this is offset by discovery surplus)
2. On his buy he was buying in the discovery (higher liquidity) and on his sell he was selling in the anchor (lower liquidity)
4. Slide moves the discovery liquidity back down so that Bob is able to buy the bAssets back at a lower average price, due to higher liquidity at a lower price.
5. Bob can pay back the bAssets he borrowed from Alice while the bAsset price is lower than when he borrowed, representing a profitable short.
Bob has made a guaranteed profit of the delta on this short.

Recommendation

Consider restructuring the way liquidity is managed in the Baseline system, such that there are no immediate large liquidity shifts which can be arbitraged in this way.

Resolution

Baseline Team: In a future iteration a gradual liquidity shift will be implemented.

H-09 | Deleverage Can Break Through Floor Tick

Category	Severity	Location	Status
Logical Error	High	CreditFacility.sol: 324	Resolved

Description

When deleveraging an account, the floor tick can be breached because the account's borrowed reserves have not yet been added to the floor tick, therefore the capacity cannot handle the flash swap — which assumes there is enough liquidity to sell the bAsset collateral before the borrowed reserves have been re-added.

Recommendation

Consider adding a requirement to the deleverage function that the existing liquidity structure, not including the virtual floor liquidity, can comfortably handle the sell of the bAsset collateral. Otherwise remove the deleverage feature and require users to deleverage manually, or implement a periphery contract which performs naive deleverages, without flash swapping, on behalf of the user.

Resolution

Baseline Team: The issue was resolved in PR#files.

H-10 | Borrows In The Floor Invalidate Baseline Value

Category	Severity	Location	Status
DoS	High	Global	Resolved

Description

Upon borrowing reserves are transferred out of the floor position and are accounted for in the virtual reserves. When a borrow occurs within the floor position, the reserves removed from the floor will not contribute towards the floor liquidity, therefore not requiring additional reserves to enter the floor upon swaps which move the price upwards out of the floor.

As a result, when reserves are moved into the virtual reserves and price moves upwards out of the floor, the virtual floor reserves are increasingly stretched wider across the floor. As a result the virtual floor reserves offer a smaller amount of liquidity and capacity as the price increases.

This can lead to an invalidation of the capacity invariant of the system, which ultimately invalidates the baseline value of the bAsset.

Recommendation

Implement validation in the `_leverage` function such that if the price were to rise to the upper tick of the floor and the capacity invariant would be invalidated, the borrow reverts. Be sure that this validation ignores any capacity gain that would come as a result of additional reserves in the Floor as price rises.

Additional reserves that would enter the floor as price increases must be ignored as they can mask an invalidation of the capacity invariant that would occur at an intermediate tick before the upper tick of the Floor.

Resolution

Baseline Team: The recommendation was implemented in commit 102c7a.

H-11 | Shorts Profit Because Of Fixed Anchor Width

Category	Severity	Location	Status
Gaming	High	Global	Acknowledged

Description PoC

In the updated Baseline V2 system the anchor is now limited to a distinct maximum width. This introduces an arbitrage whereby shorts can make a guaranteed profit by moving price into the floor, over the liquidity gap between the anchor and floor, rebalancing liquidity to fill in the gap, and buying bAsset tokens back at a lower price as the liquidity for the discovery has filled in the previous gap in the liquidity structure.

Recommendation

Consider removing the maximum width to reduce the severity of this arbitrage. Otherwise be aware of this potential gaming and consider reducing the discovery liquidity further when it moves in to fill a previous gap — thereby reducing the profitability of this manipulation.

Resolution

Baseline Team: Acknowledged.

H-12 | Anchor Liquidity Can Be Removed

Category	Severity	Location	Status
Logical Error	High	MarketMaking.sol: 305	Acknowledged

Description

In the slide function the anchor position is allocated based on the reserves of the previous anchor. However the previous anchor may have zero reserves, since it is capped to a maximum width, and the price is able to go below the anchor, between the gap if another actor creates an outside position.

As a result, the slide function will read that the anchor has 0 reserves and assign the new anchor to have 0 reserves and thus 0 liquidity.

Recommendation

Consider adopting a liquidity based approach to assigning the anchor position in slide rather than a reserves based approach.

Resolution

Baseline Team: Acknowledged.

M-01 | Launch Allows For Anchor Larger Than Target Width

Category	Severity	Location	Status
Logical Error	Medium	BaselineInit.sol: 183	Resolved

Description

In the launch function the anchor position can be more than 10 tick spacings wide as the setTicks call for the anchor position assigns the lower tick as the floor tick + one tick spacing, and the upper tick as the even tick spacing ahead of the active tick.

Recommendation

Limit the anchor position to have a lower of max(_floorTickL + T_S, activeTS - ANCHOR_WIDTH*).*

Resolution

Baseline Team: The issue was resolved in PR#92.

M-02 | Floor Reserves Decrease After Sweep

Category	Severity	Location	Status
Logical Error	Medium	MarketMaking.sol	Acknowledged

Description

In the `sweep` function the liquidity for the anchor position is maintained, however the anchor position moves higher. Therefore more reserves will be required to maintain the same liquidity for the anchor position.

Therefore if there is not enough profit from the discovery position to overshadow this discrepancy, reserves will be taken from the floor to sustain the anchor liquidity while moving the anchor up.
This may be unexpected as it can unnecessarily reduce capacity by moving reserves up from the floor position to the anchor.

Recommendation

Consider implementing the sweep function such that it maintains the reserves of the anchor position rather than the liquidity of the anchor position with the implementation listed below.
However if using this implementation be aware that this has a trade off, where the liquidity of the anchor position can now reduce upon sweeping, though by a trivial amount.

It may ultimately be fine to acknowledge this issue and keep the existing implementation as a scenario which causes more than a 10,000 wei liquidity difference has not been identified.

Resolution

Baseline Team: Acknowledged.

M-03 | Swap Fees Included In Circulating Supply

Category	Severity	Location	Status
Logical Error	Medium	MarketMaking.sol	Acknowledged

Description

The protocol deploys liquidity to Uniswap and earns fees in both reserve token and `bAsset`. The protocol owned `bAssets` and earned `bAsset` fees are not intended to be part of the circulating supply.

However, swap fees in `bAsset` are calculated as a part of the circulating supply during `bump`, `sweep` and `slide`. The reason of this is circulating assets is calculated with `bAssetsCirculating = BPOOL.totalSupply() - BPOOL.balanceOf(address(BPOOL));`.

Previously, `bAsset` fees were in the BPOOL and they were excluded while subtracting the balance of BPOOL. But after remediations, those fees are in the `MarketMaking` contract and not excluded. The newly issued tokens during `bump` is calculated based on the `bAssetsCirculating`, and more tokens will be issued due to this.

Recommendation

Consider subtracting the fees stored in the `marketMaking` contract while calculating `bAssetsCirculating`.

Resolution

Baseline Team: Acknowledged.

M-04 | Interest Free Credits Intra-Day

Category	Severity	Location	Status
Gaming	Medium	TimeslotLib.sol: 8	Resolved

Description

The `getTimeslot` function returns the unix timestamp at the end of the day as `today`, and this is used while calculating daily interests during credit borrowing.
For example, when the current time is `14:05:00`, today is considered as `23:59:59`. Because of `today` is considered as the end of the day, the time between 23:59:59 and the exact borrow time are interest free.

If a user gets a credit on June 1st at `00:00:01` for only 1 day:

• Today is June 1st `23:59:59`
• Credit end time is June 2nd `23:59:59`

User will basically get 2 days credit by paying only 1 day interest.

Recommendation

Consider adding a day to the `newExpiry_` when first initializing a loan to account for the remainder of the current day. Otherwise, clearly document this behavior to users.

Resolution

Baseline Team: The issue was resolved in commit 59118f1.

M-05 | Extra Interest Charged When Extending Credit

Category	Severity	Location	Status
Logical Error	Medium	CreditFacility.sol: 482	Acknowledged

Description

The remediation of M-H-03 is performed via adding current day to remaining days when a user extends their credit. However, the parent finding was only causing problem when a new credit is issued with `0` added days in the last day.

Current remediation charges additional one day credit when users extend their credit regardless of when the extension is performed.

Recommendation

Consider not allowing users to borrow more in the last day with `0` added days to fix the parent finding, rather than adding 1 day during every credit extension.

Otherwise, clearly document this behavior to users.

Resolution

Baseline Team: Acknowledged.

M-06 | Decrease Position Prevented

Category	Severity	Location	Status
Logical Error	Medium	CreditFacility.sol	Acknowledged

Description

The anchor and floor positions can be disjoint, and price may be between the two positions. When a user attempts to `decreasePosition`, the floor reserves will be removed before the swap.

If price is between the two positions, Uniswap will revert during the swap since there is no liquidity below the anchor as the floor was removed temporarily. Consequently, the user will be unable to `decreasePosition` and unwind their leverage.

Recommendation

Sliding will correct this case, monitor such situations and slide as necessary.

Resolution

Baseline Team: Acknowledged.

L-01 | getLiquidityForReserves Edge Case Not Handled

Category	Severity	Location	Status
Logical Error	Low	BPOOL.v1.sol: 302	Resolved

Description

When the current price at `sqrtPriceA` is below the `_sqrtPriceL`, the `getLiquidityForReserves` function will misrepresent the result as spreading the specified `_reserves` amount across the range from `[sqrtPriceA, _sqrtPriceL]` when in fact the position at `[_sqrtPriceL, _sqrtPriceU]` has no reserves in it. While currently no usage of the getLiquidityForReserves function would be prone to this bug, any future use-case is at risk.

Recommendation

Return 0 from `getLiquidityForReserves` when the `sqrtPriceA` is less than the `_sqrtPriceL`.

Resolution

Baseline Team: The issue was resolved in PR#93.

L-02 | Incorrect Comment

Category	Severity	Location	Status
Documentation	Low	BPOOL.v1.sol: 213	Resolved

Description

In the `setTicks` function the comment on line 213 states that the function is going to “calculate the corresponding sqrt prices for the tick boundaries and save them”, however there is no computation of the sqrt prices in the `setTicks` function.

Recommendation

Update this comment to reflect what the `setTicks` function does.

Resolution

Baseline Team: Resolved.

L-03 | Superfluous Capacity Calculation

Category	Severity	Location	Status
Optimization	Low	BPOOL.v1.sol: 343	Resolved

Description

In the `getCapacityForLiquidity` function the capacity is calculated using the Uniswap V3 periphery `getAmount0ForLiquidity` function if the `sqrtPriceA >= _sqrtPriceL`, however if the `sqrtPriceA == _sqrtPriceL` the result is trivially zero.

Recommendation

Do not waste gas to compute the `sqrtPriceA == _sqrtPriceL` case and alter the condition in the `getCapacityForLiquidity` to be a strict greater than comparison of `sqrtPriceA > _sqrtPriceL`.

Resolution

Baseline Team: The issue was resolved in PR#files#diff-3579424c9b33337cc1a4bf4454c2c5a502df76dc22c5110b8767b5f976fee172R312-R314.

L-04 | Unused LIQUIDITY_THICKNESS Variable

Category	Severity	Location	Status
Optimization	Low	MarketMaking.sol: 109	Resolved

Description

In the MarketMaking file the `LIQ_THICKNESS` immutable variable is assigned in the constructor yet never utilized.

Recommendation

Remove the unnecessary `LIQ_THICKNESS` variable.

Resolution

Baseline Team: The issue was resolved in PR#files#diff-3579424c9b33337cc1a4bf4454c2c5a502df76dc22c5110b8767b5f976fee172R312-R314.

L-05 | Potentially Dangerous liquidityToRemove

Category	Severity	Location	Status
Best Practices	Low	BPOOL.v1.sol: 219	Resolved

Description

In the `removeAllFrom` function the `liquidityToRemove` amount is always assigned to the `currentLiquidity` which comes from the `getLiquidity` mapping entry. In the event that the liquidity in the V3 pool is less than the entry in the mapping this will cause an underflow revert and DoS many features of the protocol.

While there is currently no identified scenario where the `getLiquidity` mapping will disagree with the liquidity of the V3 position, it is safer to only assign the `liquidityToRemove` to the current liquidity in the case where `liquidityToRemove > currentLiquidity` and that liquidity amount is guaranteed to be deployed.

Recommendation

Consider moving the `liquidityToRemove = currentLiquidity` assignment inside of the `if (liquidityToRemove > currentLiquidity)` case.

Resolution

Baseline Team: The issue was resolved in PR#files#diff-3579424c9b33337cc1a4bf4454c2c5a502df76dc22c5110b8767b5f976fee172R312-R314.

L-06 | Lack Of safeTransfer

Category	Severity	Location	Status
Best Practices	Low	CreditFacility.sol: 236	Resolved

Description

In the `_leverage` function, when leveraging the reserve tokens are sent via transfer without checking the return values. However the reserve token may potentially be a token which chooses to silently return false upon failure to transfer.

Recommendation

Use `safeTransfer` to transfer reserve tokens in the `_leverage` function.

Resolution

Baseline Team: The issue was resolved in commit 062330e.

L-07 | Outdated Comment

Category	Severity	Location	Status
Documentation	Low	MarketMaking.sol: 328	Resolved

Description

In the slide function the comment on line 328 mentions that the function “caps the new discovery liquidity to the old discovery liquidity”, however this is no longer the case.

Recommendation

Remove the outdated comment.

Resolution

Baseline Team: The issue was resolved in PR#files#diff-3579424c9b33337cc1a4bf4454c2c5a502df76dc22c5110b8767b5f976fee172R312-R314.

L-08 | Unused State Variables

Category	Severity	Location	Status
Superfluous Code	Low	MarketMaking.sol: 72, 76	Resolved

Description

`MAX_LIQ_PREMIUM` and `LIQ_THICKNESS` are defined as state variables and assigned in the constructor.

These values should've been used while calculating anchor and discovery liquidity. However, they are never used in any of the contracts.

Recommendation

Ensure that these variables are used when necessary or consider removing them.

Resolution

Baseline Team: The issue was resolved in commit 662962a.

L-09 | Inaccurate Return Value Name

Category	Severity	Location	Status
Best Practices	Low	CreditFacility.sol: 344	Resolved

Description

Function `_swapExactOut()` returns `uint256 amountOut`, although `router.exactOutputSingle` actually returns the `amountIn` necessary to achieve the exact output.

Recommendation

Change `uint256 amountOut` to `uint256 amountIn`

Resolution

Baseline Team: The issue was resolved in commit e9d33fc.

L-10 | Repay Credit Delta Does Not Match Reserves Delta

Category	Severity	Location	Status
Documentation	Low	CreditFacility.sol	Resolved

Description

During a repay it is possible for external liquidity to be included in the `getLiquidity` mapping entry. This is because on the repay `BPOOL.addReservesTo(Range.FLOOR, _repayment);` is called which will store the existing liquidity at the tick range.

Consequently, the invariant that the change in credit after the repay matches the change in the floor reserves does not hold.

Recommendation

Consider removing this vector by removing floor liquidity and adding it back during repayment.

Resolution

Baseline Team: The issue was resolved in commit e9d33fc.

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