GA GUARDIAN

Baseline

MM Looping #2

Security Assessment

February 25th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, Osman Ozdemir,

Mark Jonathas, 0xCiphky, Anonymous Auditor

Client Firm Baseline

Final Report Date February 25, 2025

Audit Summary

Baseline engaged Guardian to review the security of its market making looping updates. From the 27th of January to the 3rd 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 2 High/Critical issues were uncovered and promptly remediated 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: 1dc6ca6a06ae44b5ddae5be537c902db764608d7 Final commit: a8c321545c8505873e57a4350dbbd0f74372928d

Audit Summary

Delivery Date	February 25, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
High	1	0	0	0	1	0
Medium	3	0	0	3	0	0
Low	13	0	0	5	0	8

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
C-01	Mishandled Donated Liquidity Breaks Rebalance	DoS	Critical	Resolved
H-01	Guaranteed Profit By Rebalancing	Gaming	High	Partially Resolved
M-01	Mismatching Baseline Values	Logical Error	Medium	Acknowledged
M-02	updateTicks Should Use Previous Liquidity	Logical Error	Medium	Acknowledged
M-03	DISCOVERY_LENGTH Hardcoded For 1% Fee Pools	Logical Error	Medium	Acknowledged
L-01	Unnecessary Permission Request	Superfluous Code	Low	Resolved
L-02	Unused Error	Superfluous Code	Low	Resolved
L-03	Unused Tick Bounds In canRebalance	Superfluous Code	Low	Resolved
L-04	Active Tick Is Not Checked During Configuration	Validation	Low	Resolved
L-05	Incorrect Comment	Documentation	Low	Resolved
L-06	Unused Import	Superfluous Code	Low	Resolved
L-07	Liquidity Donations Not Tracked In New Ranges	Logical Error	Low	Acknowledged
L-08	Liquidity Position DOS In Extreme Ticks	DoS	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-09	Unecessary Min()	Superfluous Code	Low	Resolved
L-10	Tick And sqrtPrice Misalignment Side Effects	Warning	Low	Acknowledged
L-11	Missing Anchor Range	Warning	Low	Acknowledged
L-12	Remove Console Import	Superfluous Code	Low	Resolved
L-13	getCirculatingSupply Incorrect For Time	Logical Error	Low	Acknowledged

C-01 | Mishandled Donated Liquidity Breaks Rebalance

Category	Severity	Location	Status
DoS	● Critical	BPOOL.v1.sol: 209	Resolved

Description PoC

The `removeAllFrom` function in the `BPOOLv1` contract is intended to handle any unexpected liquidity (i.e., donations) by removing it from the position and then accounting for both the liquidity amount and its fees in the total `bAssetFees_`, which is subsequently sent to the fee recipient.

However, the current implementation does not behave as intended. Instead, the donated amount is burned in the internal `removeLiquidity` function while that same burned amount is also counted in `bAssetFees`.

This discrepancy causes the `MarketMaking` contract's `bAsset` balance to be lower than the total `bAssetFees_`, leading to a revert when fees are transferred.

Because of this revert, the rebalance function fails to execute, preventing the protocol from adjusting its liquidity to changing market conditions. Additionally, a malicious actor could intentionally exploit this flaw by donating, causing a DOS to the rebalance function.

Recommendation

Modify the implementation so that the donated liquidity is not burned and is correctly accounted for and transferred to the fee recipient.

Resolution

Baseline Team: Resolved.

H-01 | Guaranteed Profit By Rebalancing

Category	Severity	Location	Status
Gaming	High	MarketMaking.sol	Partially Resolved

Description PoC

The `rebalance` function removes the protocol-owned liquidity, updates ticks, and redeploys liquidity using new tick ranges. The `_updateTicks` logic determines the `anchorTick` based on whether the anchor liquidity is higher or lower than the discovery liquidity.

There are immediate arbitrage opportunities by using the rebalance functionality. When the anchor liquidity is higher than the discovery liquidity:
• The upper anchor tick is below the active tick, and the current price is within the discovery range.
• User can buy a large amount of `bToken`, pushing the price even higher.
• Call `rebalance`, which updates ticks and range liquidities.
• Sell the same amount of `bToken` at a higher average price due to higher anchor liquidity.

A similar arbitrage opportunity can occur in the opposite direction as well. When anchor liquidity is lower than discovery liquidity, the user can sell, rebalance, and buy back. This time, the user sells in a low-liquidity environment and buys back in a high-liquidity environment.

Recommendation

Consider rate limiting the amount of ticks that can be dropped at a time to limit the scale of this arbitrage vector.

Resolution

Baseline Team: Partially Resolved by rate limiting the arbitrage.

M-01 | Mismatching Baseline Values

Category	Severity	Location	Status
Logical Error	Medium	BPOOL.sol: 270	Acknowledged

Description

After the updates, the `blv` is calculated based on the upper floor tick in the `MarketMaking`, `CreditFacility`, and `LoopFacility` contracts. However, in the BPOOL contract, it is still calculated using the lower floor tick.

The `BaselineInit.launch` function uses the baseline value from the BPOOL contract when calculating capacity, which creates a discrepancy.

Recommendation

Update the `getBaselineValue` function in BPOOL contract.

Resolution

Baseline Team: Acknowledged.

M-02 | updateTicks Should Use Previous Liquidity

Category	Severity	Location	Status
Logical Error	Medium	MarketMaking.sol	Acknowledged

Description

In the `_updateTicks` function of the `MarketMaking` policy there is logic to alter the upper tick of the anchor position based upon the liquidity of the Discovery position relative to the Anchor position liquidity.

This is done to ideally prevent `bAsset` supply being minted in excess of the Discovery position liquidity in the range above the price in the Anchor position.

This reduces the magnitude of an arbitrage opportunity that would arise from selling through the Discovery range into the Anchor range and benefitting from the increased liquidity due to a higher leverage of the Anchor.

However in such an arbitrage scenario, the liquidity that the Anchor position should be compared against is the liquidity of the previous Discovery position rather than the liquidity of the new Discovery position.

This is because the old Discovery position is the one which is sold through to reach the new Anchor range and thus trigger the rebalance and therefore is the liquidity which the Arbitrage economics are based upon.

As a result the Anchor range upper tick handling should consider the liquidity of the old Discovery position rather than the current result of the `_getThresholdLiquidity` function, which will be the new Discovery position liquidity.

Recommendation

Consider comparing the predicted anchor liquidity against the minimum of both the result of the `_getThresholdLiquidity` and the old Discovery position liquidity to be the most conservative in limiting the arbitrage opportunities from selling through the Discovery position.

Resolution

Baseline Team: Acknowledged.

M-03 | DISCOVERY_LENGTH Hardcoded For 1% Fee Pools

Category	Severity	Location	Status
Logical Error	Medium	MarketMaking.sol: 58	Acknowledged

Description

The `DISCOVERY_LENGTH` variable in the `MarketMaking` contract is intended to represent 30 tick spacings, as indicated by the comment.

However, its current implementation as 30 x 200 only aligns with 1% fee pools. This means the calculation will be incorrect if applied to pools with different fee tiers.

Recommendation

If the protocol intends to only use 1% fee pools, no changes are needed. Otherwise, it should obtain the correct tick spacing from the `BPOOLv1` contract instead.

Resolution

Baseline Team: Acknowledged.

L-01 | Unnecessary Permission Request

Category	Severity	Location	Status
Superfluous Code	Low	MarketMaking.sol: 144	Resolved

Description

The `MarketMaking` contract requests permission for `BPOOL.mint` function. However, this function is not called from `MarketMaking` after updates, and the permission request can be removed.

Recommendation

Consider removing unnecessary permission request.

Resolution

Baseline Team: Resolved.

L-02 | Unused Error

Category	Severity	Location	Status
Superfluous Code	Low	MarketMaking.sol: 46	Resolved

Description

`NotOwner` error in the `MarketMaking` contract is defined but never used.

Recommendation

Remove unused error.

Resolution

Baseline Team: Resolved.

L-03 | Unused Tick Bounds In canRebalance

Category	Severity	Location	Status
Superfluous Code	Low	MarketMaking.sol: 167	Resolved

Description

The `canRebalance` function retrieves the anchor range bounds using the `_getTSBounds` function. However, these ticks are not utilized in `canRebalance`, as `rebalanceTicks` are used to determine price movement.

Recommendation

Consider removing unused ticks and the `_getTSBounds` call.

Resolution

Baseline Team: Resolved.

L-04 | Active Tick Is Not Checked During Configuration

Category	Severity	Location	Status
Validation	Low	MarketMaking.sol: 118	Resolved

Description

In the previous version of the contract, the active tick was required to be within the anchor range during configuration.

Currently, the `MarketMaking.configureDependencies` function still retrieves the lower and upper ticks of the anchor range.

However, unlike before, these ticks are no longer used for comparison against the active tick and remain unutilized.

Recommendation

If the active tick must be within the anchor range, compare it against the anchor range ticks. Otherwise, remove the `BPOOL.getTicks(Range.ANCHOR)` call from the function.

Resolution

Baseline Team: Resolved.

L-05 | Incorrect Comment

Category	Severity	Location	Status
Documentation	Low	MarketMaking.sol: 59	Resolved

Description

`BUMPABLE_PREMIUM` is set to 1500, with a comment stating "1500 tick spacings". However, this value represents only 1500 ticks, not 1500 tick spacings. Based on the current setup, it corresponds to 7.5 tick spacings.

Recommendation

Update to comment to 1500 ticks.

Resolution

Baseline Team: Resolved.

L-06 | Unused Import

Category	Severity	Location	Status
Superfluous Code	Low	MarketMaking.sol: 7	Resolved

Description

`SafeCastLib` library is imported in `MarketMaking` contract but never used.

Recommendation

Consider removing unused imports.

Resolution

Baseline Team: Resolved.

L-07 | Liquidity Donations Not Tracked In New Ranges

Category	Severity	Location	Status
Logical Error	Low	MarketMaking.sol: 307	Acknowledged

Description

The rebalance process removes liquidity from all existing ranges, checks for donations during this step, and accounts for them as fees before updating the ranges and re-adding liquidity.

However, when liquidity is added back to the new ranges, previously donated amounts in these new ranges are not accounted for.

As a result, in `_deployLiquidity`, if the discovery range had prior donations, the buffer amount and consequently the `anchorReserves` will differ from the values in `_updateTicks`, which were used to adjust the next rebalance ticks and `anchorTick`.

This discrepancy can lead to imbalances in the rebalancing logic, introducing inefficiencies in the system. For example, `_updateTicks` ensures that the anchor does not include the current price when anchor liquidity exceeds discovery liquidity.

However, a prior donation could cause this condition to be met, resulting in an extra minted supply and potentially creating an arbitrage opportunity for users, though its profitability may be limited.

Recommendation

Remove any unexpected extra liquidity as fees before adding liquidity to the new ranges to ensure consistency in rebalancing calculations.

Resolution

Baseline Team: Acknowledged.

L-08 | Liquidity Position DOS In Extreme Ticks

Category	Severity	Location	Status
DoS	Low	MarketMaking.sol, Tick.sol	Acknowledged

Description

In very extreme ticks such as -557658 an attacker would only need 300 ETH to fill all available liquidity (`liquidityGross`). This would make it impossible to rebalance a position in this range.

Therefore, if the initial active tick is around this tick, an attacker could prevent BVL from ever increasing. The same issue can occur in higher ticks for `bAsset`, where a user can buy many `bAssets` when they have a low price and DOS a liquidity range in high ticks.

Making a barrie on how high discovery range can rise before hitting the maxed out liquidity. We would need approximately 1 billion `bAssets` to DOS the liquidity range between tick 317273 and 317273 - 200 for example.

Recommendation

We recommend the team to be aware that having a very low `INITIAL_ACTIVE_TICK` (such as -557658) would make it possible to either DOS a nearby liquidity position using ETH reserves or make it possible for someone to gather enough `bAssets` to DOS a liquidity range in the higher tick ranges such as 317273, preventing the price to go further up.

Ultimately, putting a hardcoded limitation on the minimum `INITIAL_ACTIVE_TICK` would prevent this issue.

Resolution

Baseline Team: Acknowledged.

L-09 | Unecessary Min()

Category	Severity	Location	Status
Superfluous Code	Low	MarketMaking.sol	Resolved

Description

The `if` statement on line 414 already implies that `anchorReserves_ > totalReserves - _getVirtualReserves()`. Therefore the use of `min()` is redundant.

Recommendation

We recommend refactoring line 415 by removing `min()` and setting `anchorReserves_` to the difference:

```
anchorReserves_ =  totalReserves - _getVirtualReserves());
```

Resolution

Baseline Team: Resolved.

L-10 | Tick And sqrtPrice Misalignment Side Effects

Category	Severity	Location	Status
Warning	Low	MarketMaking.sol	Acknowledged

Description

When swapping `zeroToOne` an edge scenario makes it so active tick and `sqrtPrice` can be misaligned, with tick being one less than what is should.

This could lead to rebalancing positions based on the wrong tick value. Specifically, using one tick lower than what it should (in relation to `sqrtPrice`). This ultimately makes it so the positions would be assigned to a lower range than what it should.

Although we could not find a scenario where this is harmful for the protocol, we believe it has potential to do so in unforeseen circumstances or with further development.

Recommendation

We recommend the Baseline team to be mindful of this scenario as they implement new features or change the current ones.

Resolution

Baseline Team: Acknowledged.

L-11 | Missing Anchor Range

Category	Severity	Location	Status
Warning	Low	MarketMaking.sol: 285	Acknowledged

Description

When resetting the ticks for the ranges, it is possible for the anchor range to have the same value in the upper tick and lower tick. `addReservesTo()` will prevent a revert from happening by returning early, but this will lead to no liquidity deployed for the anchor range.

Recommendation

Consider shifting the upper and lower ticks if they are equal, so that the anchor range can be deployed.

Resolution

Baseline Team: Acknowledged.

L-12 | Remove Console Import

Category	Severity	Location	Status
Superfluous Code	Low	BPOOL.v1.sol: 17 & MarketMaking.sol: 23	Resolved

Description

`BPOOL.v1.sol` and `MarketMaking` imports the `console2` library. Console logging can be removed for production since it is unnecessary.

Recommendation

Remove the imports of the `console2` library.

Resolution

Baseline Team: Resolved.

L-13 | getCirculatingSupply Incorrect For Time

Category	Severity	Location	Status
Logical Error	Low	MarketMaking.sol	Acknowledged

Description

The `getCirculatingSupply` view function in the `MarketMaking` contract does not account for collateral in the Loops facility which is burned with time passing.

However the corresponding `getTotalCapacity` function will account for the capacity change with respect to time resulting from the `LOOPS.totalDebt()`.

This may mislead users and integrators and increases the risk of this function causing bugs in the future due to this undocumented behavior.

Recommendation

Consider adjusting the `getCirculatingSupply` function to account for the collateral in the Loops facility which is yet to be burnt.

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