

**GA GUARDIAN**

# Baseline Markets V2

**Security Assessment**

June 17th, 2024



# Summary

**Audit Firm** Guardian

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


**Client Firm** Baseline Markets

**Final Report Date** June 17, 2024

## Audit Summary

Baseline Markets engaged Guardian to review the security of its concentrated liquidity protocol, supporting a baseline value for its YES token. From the 29th of April to the 9th of May, 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

Invariants Assessed ..... 7

Findings & Resolutions ..... 9

## Addendum

Disclaimer ..... 57

About Guardian Audits ..... 58

# Project Overview

## Project Summary

Project Name	Baseline
Language	Solidity
Codebase	<a href="https://github.com/0xBaseline/baseline-v2">https://github.com/0xBaseline/baseline-v2</a>
Commit(s)	89327ab764af1c3512b09533987003636886e7c6

## Audit Summary

Delivery Date	June 17, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

## Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	2	0	0	0	1	1
● High	8	0	0	0	0	8
● Medium	15	0	0	4	0	11
● Low	19	0	0	3	4	12

# Audit Scope & Methodology

## Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● 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.

# Invariants Assessed

During Guardian’s review of the Baseline V2 contracts, fuzz-testing with [Echidna](#) was performed on the protocol’s main functions. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Tested	Passed	Run Count
<u>BUMP-01</u>	bump() should return true if successful	✓	✓	10,000,000+
<u>BUMP-02</u>	Floor lower tick should increase by TICK_SPACING	✓	✓	10,000,000+
<u>BUMP-03</u>	Anchor lower tick should increase by TICK_SPACING	✓	✓	10,000,000+
<u>BUMP-04</u>	Anchor upper tick should not change	✓	✓	10,000,000+
<u>BUMP-05</u>	Discovery lower tick should not change	✓	✓	10,000,000+
<u>BUMP-06</u>	Discovery upper tick should not change	✓	✓	10,000,000+
<u>BUMP-07</u>	Balance of EMISSIONS_RECIPIENT should increase if there is circulating supply	✓	✓	10,000,000+
<u>BUMP-08</u>	Anchor liquidity should not change	✓	✓	10,000,000+
<u>BUMP-09</u>	Discovery liquidity should not change	✓	✓	10,000,000+
<u>BUMP-10</u>	Floor position reserve balance should not decrease	✓	✓	10,000,000+

# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>BUMP-11</u>	Anchor position reserve balance should not increase	✓	✓	10,000,000+
<u>BUMP-12</u>	System is solvent before bump call	✓	✓	10,000,000+
<u>BUMP-13</u>	System is solvent after bump call	✓	✓	10,000,000+
<u>BUMP-14</u>	BPOOL should not increase in bAssets	✓	✓	10,000,000+
<u>BUMP-15</u>	BPOOL should be cleared of reserves	✓	✓	10,000,000+
<u>BUMP-16</u>	Bump should not revert besides insolvency and failed canBump	✓	✗	10,000,000+
<u>SWEEP-01</u>	Should make no state changes if it returns false	✓	✓	10,000,000+
<u>SWEEP-02</u>	Sweep is callable when active tick moves up by more than one TICK_SPACING and is less than DISCOVERY upper tick (within Baseline Positions)	✓	✓	10,000,000+
<u>SWEEP-03</u>	Active tick should be in anchor range after sweep rebalance operation	✓	✓	10,000,000+
<u>SWEEP-04</u>	Discovery lower tick should be greater than activeTick after sweep	✓	✓	10,000,000+
<u>SWEEP-05</u>	Active Tick Matches Checkpoint Tick	✓	✓	10,000,000+
<u>SWEEP-06</u>	Active Tick Is Above Floor	✓	✓	10,000,000+
<u>SWEEP-07</u>	Anchor range should not shrink	✓	✓	10,000,000+
<u>SWEEP-08</u>	Anchor liquidity should not decrease	✓	✗	10,000,000+
<u>SWEEP-09</u>	Floor liquidity should not decrease	✓	✓	10,000,000+



# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>SWEEP-10</u>	Discovery liquidity should not exceed max ratio	✓	✗	10,000,000+
<u>SWEEP-11</u>	BPOOL bAsset balance is zero	✓	✓	10,000,000+
<u>SWEEP-12</u>	BPOOL reserve balance is zero	✓	✓	10,000,000+
<u>SWEEP-13</u>	After sweep no bAssets in floor	✓	✓	10,000,000+
<u>SWEEP-14</u>	After sweep no reserves in discovery	✓	✓	10,000,000+
<u>SWEEP-15</u>	Anchor reserves increased	✓	✓	10,000,000+
<u>SWEEP-16</u>	Anchor reserves non-zero	✓	✓	10,000,000+
<u>SWEEP-17</u>	Anchor bAssets non-zero	✓	✓	10,000,000+
<u>SWEEP-18</u>	Discovery bAssets non-zero	✓	✓	10,000,000+
<u>SWEEP-19</u>	Floor Reserves increased	✓	✓	10,000,000+
<u>SWEEP-20</u>	Solvency Is Maintained Before Sweep	✓	✓	10,000,000+
<u>SWEEP-21</u>	Solvency Is Maintained After Sweep	✓	✓	10,000,000+
<u>SWEEP-22</u>	Capacity is greater after sweep	✓	✗	10,000,000+
<u>SWEEP-23</u>	Sweep should not revert beyond Insolvent error	✓	✗	10,000,000+

# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>SLIDE-01</u>	Should make no state changes if it returns false	✓	✓	10,000,000+
<u>SLIDE-02</u>	Slide is callable when active tick moves down by more than one TICK_SPACING from checkpoint tick and is less than DISCOVERY lower tick (You can't slide in DISCOVERY range)	✓	✓	10,000,000+
<u>SLIDE-03</u>	Anchor should shrink on slide	✓	✓	10,000,000+
<u>SLIDE-04</u>	Discovery range size should remain the same	✓	✓	10,000,000+
<u>SLIDE-05</u>	Checkpoint tick updated to active tick	✓	✓	10,000,000+
<u>SLIDE-06</u>	Floor ticks should not change	✓	✓	10,000,000+
<u>SLIDE-07</u>	Discovery Liquidity Should Decrease	✓	✗	10,000,000+
<u>SLIDE-08</u>	Anchor liquidity should not increase	✓	✓	10,000,000+
<u>SLIDE-09</u>	Floor reserves should not decrease by a significant amount	✓	✓	10,000,000+
<u>SLIDE-10</u>	BPOOL bAsset balance should be zero	✓	✓	10,000,000+
<u>SLIDE-11</u>	BPOOL reserve balance should be zero	✓	✓	10,000,000+
<u>SLIDE-12</u>	Anchor reserves should not increases	✓	✓	10,000,000+
<u>SLIDE-13</u>	Discovery bAssets are non-zero	✓	✓	10,000,000+
<u>SLIDE-14</u>	Floor reserves are non-zero	✓	✓	10,000,000+

# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>SLIDE-15</u>	Solvency Is Maintained Before Slide	✓	✓	10,000,000+
<u>SLIDE-16</u>	Solvency Is Maintained After Slide	✓	✓	10,000,000+
<u>SLIDE-17</u>	Capacity should not decrease	✓	✓	10,000,000+
<u>SLIDE-18</u>	Circulating supply should not increase on slide by a significant amount	✓	✓	10,000,000+
<u>SLIDE-19</u>	Slide should not revert besides Insolvent error	✓	✓	10,000,000+
<u>BORROW-01</u>	Days added or account expiry must be greater than zero for a successful borrow transaction	✓	✓	10,000,000+
<u>BORROW-02</u>	New expiry return data after borrow is called should match the creditor's account details	✓	✓	10,000,000+
<u>BORROW-03</u>	The expiry must be in the future	✓	✓	10,000,000+
<u>BORROW-04</u>	Borrowers reserve balance should increase by the borrow reserveOut amount	✓	✓	10,000,000+
<u>BORROW-05</u>	Borrowers bAsset balance should decrease by the borrow collateral amount	✓	✓	10,000,000+
<u>BORROW-06</u>	Borrowers account collateral should increase by deposited collateral	✓	✓	10,000,000+
<u>BORROW-07</u>	Borrowers account credit should increase by the sum of received reserves and interest paid	✓	✓	10,000,000+
<u>BORROW-08</u>	User should always pay interest for credits	✓	✗	10,000,000+
<u>BORROW-09</u>	Credit can't be higher than collateral, before borrow operation	✓	✓	10,000,000+

# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>BORROW-10</u>	Credit can't be higher than collateral, after borrow operation	✓	✓	10,000,000+
<u>BORROW-11</u>	The totalInterestAccumulated should increase by the interest paid after every borrow	✓	✓	10,000,000+
<u>BORROW-12</u>	CreditFacility bAsset balance should be 0	✓	✓	10,000,000+
<u>BORROW-13</u>	CreditFacility reserve balance should be 0	✓	✓	10,000,000+
<u>BORROW-14</u>	BPOOL should hold no reserve in its balance	✓	✓	10,000,000+
<u>BORROW-15</u>	If the borrowed credit is paid with reserves in floor position, reserves in anchor should not change	✓	✓	10,000,000+
<u>BORROW-16</u>	If the borrowed credit is paid with reserves in floor position, reserves in discovery positions should not change	✓	✓	10,000,000+
<u>BORROW-17</u>	If the borrowed credit is paid with reserves in floor and anchor positions, reserves in discovery positions should not change	✓	✓	10,000,000+
<u>BORROW-18</u>	Borrow should not revert if the function parameters are valid and there's enough reserve to cover the credit	✓	✓	10,000,000+
<u>REPAY-01</u>	The amount of bAsset returned to user should be equal to the amount of collateral in the user's credit account	✓	✓	10,000,000+
<u>REPAY-02</u>	The amount of reserves returned by user should be equal to the amount of credit in the user's credit account	✓	✓	10,000,000+
<u>REPAY-03</u>	There should be no reserve stuck in creditFacility after repay operation	✓	✓	10,000,000+

# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>REPAY-04</u>	There should be no reserve stuck in BPOOL after repay operation	✓	✓	10,000,000+
<u>REPAY-05</u>	Credit can't be higher than collateral, before repay operation	✓	✓	10,000,000+
<u>REPAY-06</u>	Credit can't be higher than collateral, after repay operation	✓	✓	10,000,000+
<u>REPAY-07</u>	The repaid reserves should go into floor position	✓	✓	10,000,000+
<u>REPAY-08</u>	If the REPAY-ed credit is paid with reserves in floor position, reserves in anchor should not change	✓	✓	10,000,000+
<u>REPAY-09</u>	If the REPAY-ed credit is paid with reserves in floor position, reserves in discovery should not change	✓	✓	10,000,000+
<u>REPAY-10</u>	Repay should not revert given if user's account credit is greater than zero, and can transfer the required reserve amount	✓	✓	10,000,000+
<u>DFLOT-01</u>	Total collateralized should decrease	✓	✓	10,000,000+
<u>DFLOT-02</u>	defaultOutstanding operation should send bAsset to BPOOL after default	✓	✓	10,000,000+
<u>DFLOT-03</u>	Total credit issued should not change	✓	✓	10,000,000+
<u>DFLOT-04</u>	Total collateralized should not change	✓	✓	10,000,000+
<u>DFLOT-05</u>	Last defaulted timeslot should be today	✓	✓	10,000,000+
<u>DFLOT-06</u>	User's reserve balance should not change	✓	✓	10,000,000+
<u>DFLOT-07</u>	User's bAsset balance should not change	✓	✓	10,000,000+









# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>DFLOT-08</u>	Total interest accumulated should not change	✓	✓	10,000,000+
<u>DFLOT-09</u>	Floor position reserve should not change	✓	✓	10,000,000+
<u>DFLOT-10</u>	Anchor position reserve should not change	✓	✓	10,000,000+
<u>DFLOT-11</u>	Discovery position reserve should not change	✓	✓	10,000,000+
<u>DFLOT-12</u>	Credit facility reserve balance should not change	✓	✓	10,000,000+
<u>DFLOT-13</u>	Credit facility bAsset balance should not change	✓	✓	10,000,000+
<u>DFLOT-14</u>	BPOOL reserve balance should not change	✓	✓	10,000,000+
<u>DFLOT-15</u>	CREDIT reserve balance should not change	✓	✓	10,000,000+
<u>DFLOT-16</u>	CREDIT bAsset balance should not increase	✓	✓	10,000,000+
<u>DFLOT-17</u>	DefaultOutstanding should not revert	✓	✓	10,000,000+
<u>DFSF-01</u>	Account credit should be greater than zero for successful defaultSelf operation	✓	✓	10,000,000+
<u>DFSF-02</u>	DefaultSelf should set the caller's account credit to zero	✓	✓	10,000,000+
<u>DFSF-03</u>	DefaultSelf should set the caller's account collateral to zero	✓	✓	10,000,000+
<u>DFSF-04</u>	DefaultSelf should set the caller's account expiry to zero	✓	✓	10,000,000+

# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>DFSF-05</u>	DefaultSelf should reduce total credit issued by the account's credit	✓	✓	10,000,000+
<u>DFSF-06</u>	DefaultSelf should reduce total collateralized by the account's collateral	✓	✓	10,000,000+
<u>DFSF-07</u>	DefaultSelf should send defaulted bAsset to BPOOL after default	✓	✓	10,000,000+
<u>DFSF-08</u>	DefaultSelf shouldn't revert if the credit in the user's account is greater than zero	✓	✓	10,000,000+
<u>MM-01</u>	Anchor liquidity should always be thinner than discovery	✓	✓	10,000,000+
<u>MM-02</u>	Checkpoint tick should never be below the floor	✓	✓	10,000,000+
<u>MM-03</u>	Active tick should never be below the floor	✓	✓	10,000,000+
<u>MM-04</u>	Solvency invariant: total capacity should never be less than circulating supply	✓	✓	10,000,000+
<u>MM-05</u>	Discovery Liquidity Is Always Greater Than 0	✓	✓	10,000,000+
<u>MM-06</u>	Positions Ticks Always Multiple Of Tick Spacing	✓	✓	10,000,000+
<u>MM-07</u>	Verify that the liquidity in the PositionData for each range should correspond to the actual liquidity in the Uniswap V3 pool for that range.	✓	✓	10,000,000+
<u>CREDIT-01</u>	The amount of bAsset in CREDIT should match the total collateralized	✓	✓	10,000,000+
<u>CREDIT-02</u>	The lastDefaultedTimeslot should not be ahead of the current day	✓	✓	10,000,000+
<u>CREDIT-03</u>	The sum of all individual collateral amounts should equal the total total collateralized.	✓	✓	10,000,000+

# Invariants Assessed

ID	Description	Tested	Passed	Run Count
<u>CREDIT-04</u>	The sum of all individual credit amounts should equal the total credit issued.			10,000,000+
<u>CREDIT-05</u>	A user's credit account should either have both credit and collateral or neither.			10,000,000+
<u>CREDIT-06</u>	Baseline value of total collateral should be greater than credit provided			10,000,000+
<u>CREDIT-07</u>	The value of a borrower's credit should never be higher than the value of their collateral			10,000,000+



# Findings & Resolutions

ID	Title	Category	Severity	Status
<a href="#">C-01</a>	DoS By Adding Liquidity On Behalf Of BPOOL	DoS	● Critical	Partially Resolved
<a href="#">C-02</a>	Floor Inflation Allows Risk Free Shorts	Gaming	● Critical	Resolved
<a href="#">H-01</a>	Anchor Liquidity Is Incorrectly Calculated After Sweep	Logical Error	● High	Resolved
<a href="#">H-02</a>	Invalid Leverage Factor Calculation	Logical Error	● High	Resolved
<a href="#">H-03</a>	Interest Free Borrowing Due To TimeslotLib	Gaming	● High	Resolved
<a href="#">H-04</a>	Incorrect Calculation of Upper Tick in Anchor Range	Logical Error	● High	Resolved
<a href="#">H-05</a>	Migration Process Can Be DoS'd	DoS	● High	Resolved
<a href="#">H-06</a>	Interest Formula Favours Longer Credits	Logical Error	● High	Resolved
<a href="#">H-07</a>	All Gas Yields Earned On Blast Are Lost	Logical Error	● High	Resolved
<a href="#">H-08</a>	Protocol Loses Blast Points And WETH Rebasing Yields	Logical Error	● High	Resolved
<a href="#">M-01</a>	Lacking Initial Tick Validation May Brick The Protocol	DoS	● Medium	Resolved
<a href="#">M-03</a>	Anchor Liquidity Heavily Reduced	Logical Error	● Medium	Resolved
<a href="#">M-04</a>	Liquidity Premium Rounds Down To 0	Logical Error	● Medium	Resolved

# Findings & Resolutions

ID	Title	Category	Severity	Status
<a href="#">M-05</a>	Discovery Liquidity Increased Above MAX_DISCOVERY_RATIO	Logical Error	● Medium	Resolved
<a href="#">M-06</a>	Bump Reverts Due To Lack Of Reserves	DoS	● Medium	Acknowledged
<a href="#">M-07</a>	Rebalance Threshold May Prevent Sliding Operation	DoS	● Medium	Acknowledged
<a href="#">M-08</a>	Incorrect Circulating Supply Calculation	Logical Error	● Medium	Acknowledged
<a href="#">M-09</a>	Initial Deployed Liquidity Missing Crucial Validations	Validation	● Medium	Resolved
<a href="#">M-10</a>	Virtual Liquidity Lower Than Expected	Logical Error	● Medium	Resolved
<a href="#">M-11</a>	Operations Using Outdated Circulating Supply	Logical Error	● Medium	Resolved
<a href="#">M-12</a>	Arbitrage Attack After Slide Operation	Logical Error	● Medium	Acknowledged
<a href="#">M-13</a>	Attackers Can Borrow With Zero Interest	Gaming	● Medium	Resolved
<a href="#">M-14</a>	liquidityPremium Can Be A Discount	Logical Error	● Medium	Resolved
<a href="#">M-15</a>	getLeverageFactor DoS	DoS	● Medium	Resolved
<a href="#">M-16</a>	Risk Free Arbitrage Attack	DoS	● Medium	Resolved
<a href="#">L-01</a>	Slide Prevents Price To Exit The Floor	Logical Error	● Low	Acknowledged

# Findings & Resolutions

ID	Title	Category	Severity	Status
<a href="#">L-02</a>	Discovery Liquidity Increased After Sliding	Logical Error	● Low	Acknowledged
<a href="#">L-03</a>	Slide May Result In Equal Anchor And Discovery Liquidity	Logical Error	● Low	Acknowledged
<a href="#">L-04</a>	Strict Inequality For Anchor Threshold Comparison	Logical Error	● Low	Partially Resolved
<a href="#">L-05</a>	Reserve Token Should Use safeTransfer	Logical Error	● Low	Resolved
<a href="#">L-06</a>	Inconsistent Default Pattern Array Lengths	Logical Error	● Low	Resolved
<a href="#">L-07</a>	Superfluous brs Address	Superfluous Code	● Low	Resolved
<a href="#">L-08</a>	Users May Repay 0 Amount	Validation	● Low	Resolved
<a href="#">L-09</a>	Invalid Event Emission Data	Logical Error	● Low	Resolved
<a href="#">L-10</a>	Debug Code In Production	Superfluous Code	● Low	Partially Resolved
<a href="#">L-11</a>	Misleading Comments	Documentation	● Low	Resolved
<a href="#">L-12</a>	Unused Parameters	Superfluous Code	● Low	Partially Resolved
<a href="#">L-13</a>	Redundant Ternary Operators	Superfluous Code	● Low	Partially Resolved
<a href="#">L-14</a>	Defaulted Event Logging Incorrect Data	Superfluous Code	● Low	Resolved

# Findings & Resolutions

ID	Title	Category	Severity	Status
<a href="#">L-15</a>	Unnecessary Allowances	Superfluous Code	<div><div></div>Low</div>	Resolved
<a href="#">L-16</a>	Array Lengths Don't Match During Default Configuration	Logical Error	<div><div></div>Low</div>	Resolved
<a href="#">L-17</a>	Hardcoded Reserve Token Address Is Incorrect	Logical Error	<div><div></div>Low</div>	Resolved
<a href="#">L-18</a>	totalCreditIssued Includes Interest Which Has Yet To Be Paid	Logical Error	<div><div></div>Low</div>	Resolved
<a href="#">L-19</a>	Floor Tick Increment Prevents Sliding Operations	Validation	<div><div></div>Low</div>	Resolved

# C-01 | DoS By Adding Liquidity On Behalf Of BPOOL

Category	Severity	Location	Status
DoS	● Critical	MarketMaking.sol: 190	Partially Resolved

## Description

The protocol has three main functions, which are bump, sweep and slide, to maintain liquidity structure among different ranges. During bump, all liquidity is removed from floor, anchor and discovery, and the exactly previous amount of liquidity is added back to anchor and discovery.

```
(,, uint128 liquidityA) = BPOOL.removeAllFrom(Range.ANCHOR);
(,, uint128 liquidityD) = BPOOL.removeAllFrom(Range.DISCOVERY);
// ...
BPOOL.manageLiquidityFor(Range.ANCHOR, Action.ADD, liquidityA);
BPOOL.manageLiquidityFor(Range.DISCOVERY, Action.ADD, liquidityD);
```

Normally, the protocol should never have more liquidity in anchor than discovery. However, anyone can directly mint on behalf of any other user in Uniswap. An attacker can break this invariant by adding liquidity to anchor and bumping right after.

After this, the sweep will always revert due to underflow [here](#). slide and bump will also always revert regardless of the active price if the attack is done when the checkpointTick is equal to floorUpper + tickSpacing, and the protocol's liquidity structure will be completely stuck.

## Recommendation

There is no way to prevent attackers minting directly on Uniswap. To prevent this issue, keep track of the protocol owned liquidity as a separate variable and use it to determine liquidity amounts.

## Resolution

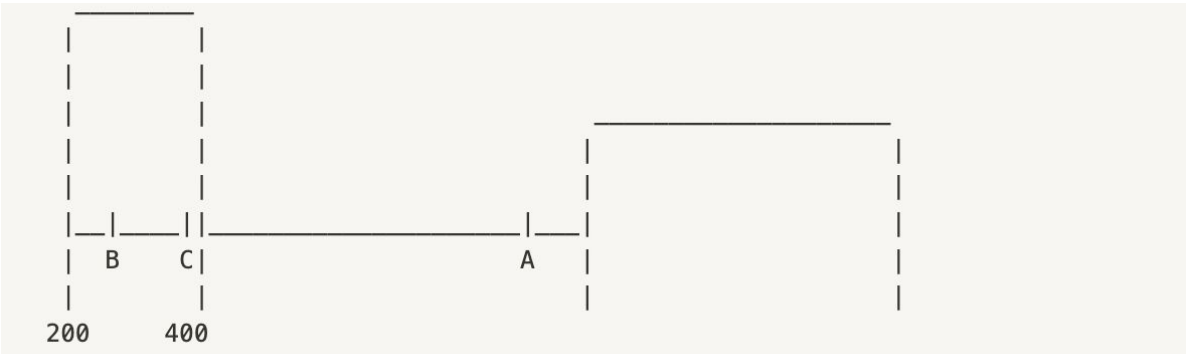
Baseline Team: The issue was resolved in [PR#48](#).

# C-02 | Floor Inflation Allows Risk Free Shorts

Category	Severity	Location	Status
Gaming	● Critical	MarketMaking.sol: 312	Resolved

## Description

In the `slide` function the amount of liquidity credited to the floor range is dependent on the balance of the `BPOOL` contract after allocating the discovery and anchor positions. Users who hold a large amount of `bAssets` can game this behavior for an immediate and significant gain in their `bAsset` amount. Consider the following scenario:



A Whale user starts at price A and sells `bAssets` to move price into the floor at price B. The user then maliciously sends reserve tokens directly to the `BPOOL` contract and triggers a `slide`. During the `slide` the reserve tokens sent this way are attributed to the liquidity of the floor. However since price is inside of the floor, these `bAssets` are "leveraged" as for the reserve tokens deployed to the floor position, there are corresponding `bAssets` which are minted to be paired.

Now using the increased "leveraged" liquidity in the floor position, the user swaps all of the remaining reserves they had received from the original sell. Only now, due to the increased liquidity of the floor, they only reach C as a final price on their buy. This way the user's average price on the buy is much lower than the average price on their sell, and they realize an immediate arbitrage profit in terms of `bAssets`, this guarantees profit on a `bAsset` short.

## Recommendation

Burn any assets in the `BPOOL` prior to removing the three positions from the liquidity pool, this way malicious actors cannot tamper with the resulting liquidity amounts in the floor position and create profitable scenarios.

## Resolution

Baseline Team: The issue was resolved in [PR#48](#).

# H-01 | Anchor Liquidity Is Incorrectly Calculated After Sweep

Category	Severity	Location	Status
Logical Error	● High	MarketMaking.sol: 219	Resolved

## Description

During sweep, discovery liquidity is rebalanced into anchor and floor liquidity. Anchor’s tick range is also increased. It was observed that anchor’s liquidity could decrease post-sweep which is undesirable.

The issue stems from the increased anchor tick range and calling `BPOOL.manageReservesFor(Range.ANCHOR, Action.ADD, newReservesA);` instead of `BPOOL.manageLiquidityFor()`.

There is no guarantee that `newReservesA` is sufficient to maintain or increase anchor’s liquidity, given the new tick range. Although `surplusReservesD` (surplus reserves in discovery) is added to `newReservesA`, this may not be sufficient as surplus could be small or even zero.

As a result, every time sweep is called anchor liquidity could decrease, leading to a very thin anchor range and poor trading conditions where price fluctuates wildly between floor and discovery.

## Recommendation

Calculate the amount of reserves needed to maintain `anchor.liquidity` then add surplus reserves before re deploying Anchor liquidity.

## Resolution

Baseline Team: The issue was resolved in [PR#55](#).

# H-02 | Invalid Leverage Factor Calculation

Category	Severity	Location	Status
Logical Error	● High	MarketMaking.sol: 314	Resolved

## Description

The leverage factor is calculated as the ratio between the totalCollateralized and the spot supply (externally owned bAssets). Whenever sweep or slide rebalances are executed, the leverage factor will act as a multiplier for the liquidityPremium.

The main issue relies on the calculation of this multiplier. The first part of the formula correctly calculates the ratio: `leverageFactor_ = totalCollateral / (_bAssetsCirculating - totalCollateral);`

But the value returned by the function is:  
`leverageFactor_ += 1e18;`

This means that if the ratio is 5, the leverage factor will be  $1e18 + 5$  instead of  $5e18$  or a 5x multiplier. The issue can also be found at [InitializeProtocol.sol#L166](#) as well.

## Recommendation

Use the FixedPointMathLib lib to correctly return the multiplier value:  
`leverageFactor_ = 1e18 + totalCollateral.divWad(_bAssetsCirculating - totalCollateral);`

## Resolution

Baseline Team: The issue was resolved in [PR#66](#).



# H-03 | Interest Free Borrowing Due To TimeslotLib

Category	Severity	Location	Status
Gaming	● High	CreditFacility.sol: 181	Resolved

## Description

Users can borrow reserves from CreditFacility using bAssets as collateral. A small interest will be paid based on the credit amount and the days added. The issue arises when users borrow more reserves with an existing credit account, without adding more days to expiry time.

The protocol will try to calculate the interestOnNewCollateral based on the daysRemaining, but this value is 0 as there won't be any days remaining during the last expiry day.

Any user borrowing during the last expiry day of the account, and does not add more days to the expiration, will effectively pay no interest for the new credit. Users might take advantage of this issue, by borrowing reserves from the FLOOR and ANCHOR, and repaying them in the same transaction to the FLOOR, and only pay gas fees.

This will cause the capacity to be increased at will, as well as reduce the ANCHOR reserves that support the current price, opening the opportunity for arbitrageurs to extract reserves from the protocol.

## Recommendation

Prevent users from borrowing more credit during the last day of expiry if they are not adding more days to account expiration.

## Resolution

Baseline Team: The issue was resolved in [PR#51](#).

# H-04 | Incorrect Calculation of Upper Tick in Anchor Range

Category	Severity	Location	Status
Logical Error	● High	BPOOL.v1.sol	Resolved

## Description

The MarketMaking policy deploys liquidity to FLOOR, ANCHOR and DISCOVERY positions when rebalancing and their tick boundaries are calculated based on the `getTickBoundaries` function. The upper tick of ANCHOR (lower tick of DISCOVERY is calculated based on `_getUpperAnchorTick` which is suppose to return the next highest initialized tick that is a multiple of `TICK_SPACING` as stated in the docs.

Although the `_getUpperAnchorTick` works with positive ticks, it does not return the correct value for negative ticks. This is due to the fact that the function is rounding the current tick down in absolute value, but negative ticks should be rounded up. When the active tick is in the FLOOR range, the ANCHOR range will disappear when the active tick is positive, but it won't when its negative.

This also creates an unexpected scenario where the `sweep` rebalance operation can be executed when the price is at ANCHOR range. This operation will only check the reserves at ANCHOR, but not the liquidity removed as `bAssets`, which will be burned.

## Recommendation

Consider refactoring the formula to return the correct value. A suggested approach is:

```
function _getUpperAnchorTick() internal view returns (int24 upperAnchorTick_) {
    if (checkpointTick % TICK_SPACING == 0) return checkpointTick + TICK_SPACING;
    upperAnchorTick_ = (checkpointTick / TICK_SPACING) * TICK_SPACING;
    if (checkpointTick > 0) {
        upperAnchorTick_ += TICK_SPACING;
    }
}
```

## Resolution

Baseline Team: The issue was resolved in [PR#49](#).

# H-05 | Migration Process Can Be DoS'd

Category	Severity	Location	Status
DoS	● High	InitializeProtocol.sol	Resolved

## Description

During the migration process, the protocol will:

1. Initialize the pool
2. Distribute spot tokens and create credits
3. Deploy liquidity to the pool

This migration is not atomic and will take some time, which means there will be a lag between the initialization of the Uniswap pool and deploying liquidity to the pool. In this period, an attacker can add liquidity to any tick he wants, perform a swap, and change the active tick. Since the pool is empty, this can be done with only a few wei.

Attacker injects a swap between steps 1 and 3 above but the migration process continues, and there is no check regarding whether the current tick is the same as INITIAL\_ACTIVE\_TICK.

- There are two possible impacts depending on which way the swap is performed by the attacker.
1. If the attacker swaps below the floor tick, the liquidity deployment will be successful, but the ratio between discovery and anchor will be enormous.
  2. If the attacker swaps way above, liquidity deployment will fail due to active tick being in the discovery range.

## Recommendation

Consider refactoring the migration process such that there is no time window for swaps to occur between the initialization of the pool and deploying liquidity to the pool.

## Resolution

Baseline Team: The issue was resolved in [PR#68](#).

# H-06 | Interest Formula Favours Longer Credits

Category	Severity	Location	Status
Logical Error	● High	CreditFacility.sol: 297	Resolved

## Description

The protocol has a credit feature and the interests for these credits are paid upfront. Interest is meant to be linear based on `INTEREST_PER_DIEM` variable and credit length.

However, interest formula favours longer credits due to an error in it. Currently:

- Yearly credit interest is  $\sim 348.6 \times$  daily interest instead of 365.
- Yearly credit interest is  $\sim 11.6 \times$  monthly interest instead of 12.

This means  $\sim 4.5\%$  discount in interest when taking yearly credits.

The protocol has two major revenue streams: LP fees and credit interests. This income is used to increase the baseline value of the token. Due to this formula favouring longer credits, protocol loses  $\sim 3\text{-}5\%$  of its expected interest revenue (*exact number will be affected by average credit length*).

## Recommendation

Update the interest formula to prevent value loss, and ensure the interest rate is fixed regardless of the credit length.

## Resolution

Baseline Team: The issue was resolved in [PR#69](#).

# H-07 | All Gas Yields Earned On Blast Are Lost

Category	Severity	Location	Status
Logical Error	● High	Global	Resolved

## Description

This protocol will be deployed on Blast, which provides a feature to claim gas fees consumed by smart contracts. Contracts can claim 50% to 100% of gas fees depending on the claim rate.

The default gas mode for contracts on Blast is Void, and all consumed gas is sent to the sequencer. For these gas fees to be claimed, smart contracts must be [configured](#) by interacting with the BLAST contract at `0x430002` address.

## Recommendation

Configure smart contracts to set gas mode `claimable`, and use this additional income to increase `blv`.

## Resolution

Baseline Team: The issue was resolved in commit [f1f9e5d](#).

# H-08 | Protocol Loses Blast Points And WETH Rebasing Yields

Category	Severity	Location	Status
Logical Error	● High	Global	Resolved

## Description

The reserve token in the protocol is Blast’s rebasing WETH token, which has automatic rebasing by default (not like the native ETH with void by default) for both EOAs and smart contracts.

Normally, these reserve tokens are not held in the Baseline contracts. They are deployed into the Thruster [pools](#), which set their WETH yield configuration as claimable in their constructor, as liquidity. So, Thruster pools earn rebasing yields from liquidity providers’ assets, and the factory owner of the pool can claim these yields.

According to [Thruster docs](#), liquidity providers earn Blast points depending on their WETH balances since they help Thruster to earn yields. Thruster says the system is automatic, but there are some checks: *“Pools with a significant amount of liquidity providers being contracts (not EOAs) need to be manually verified to ensure that the Points are claimable by the contracts”*.

BPOOL contract itself will be the biggest liquidity provider but it has no way to claim Blast points. Therefore, the Thruster protocol [will not allocate](#) these earned points to Baseline. *“...it should not be allocated to a contract address that does not support the Blast Points API as those Points then become unclaimable or transferable...”*.

This would also be a case if the protocol decides to deploy liquidity in another Uniswap fork.

## Recommendation

Use `IERC20Rebasing` interface instead of regular `ERC20` for reserve token, and then configure smart contracts in a way to be integrated with Blast points system.

## Resolution

Baseline Team: Resolved.

# M-01 | Lacking Initial Tick Validation May Brick The Protocol

Category	Severity	Location	Status
DoS	● Medium	BPOOL.v1.sol: 117	Resolved

## Description

The BPOOL module is initialized with the starting values for activeTick and floorTick. The current validation only checks that activeTick > floorTick. At this moment, checkpointTick is also initialized with the same value as activeTick.

The issue rises when initial activeTick value is not greater than floorTick + TS, as the active tick will be at the FLOOR range, ANCHOR range won't exist, and DISCOVERY range will be initialized with 0 liquidity.

Additionally, if this initial setup is created, then no marketMaking operations can be executed, even if the price start trading upwards:

- bump - checkpoint is at the FLOOR
- sweep - Panic reverts when calculating liquidityPremium as liquidityA is 0
- slide - activeTick is above checkpoint

## Recommendation

Replace the tick values validation in the initializePool with:  
require(\_initialFloorTick + TS < \_initialActiveTick, "Invalid tick values");

## Resolution

Baseline Team: The issue was resolved in [PR#67](#).

# M-03 | Anchor Liquidity Heavily Reduced

Category	Severity	Location	Status
Logical Error	● Medium	MarketMaking.sol: 289	Resolved

## Description

The `slide` operation scales down the ANCHOR liquidity using the `inverseLiquidityPremium` as follows:  
`liquidityA = uint128(uint256(liquidityA).divWad((inverseLiquidityPremium)));`

The issue arises when all (or most) of the circulating supply is collateralized, and the leverage factor is a very large number. This means the `liquidityA` can drop to unexpected low values, creating high slippage to the users currently trading.

If `slide` is triggered when the price is at the ANCHOR range (normal scenario), the operation will effectively remove most of the liquidity from the active trading range, moving the active reserves to the FLOOR range. Although the capacity of the system is still increasing, any subsequent sales will move the active tick into the FLOOR, something the protocol wants to avoid.

In order to restore the correct liquidity of the ANCHOR, price will need to trade up into DISCOVERY and receive surplus reserves. Although this might temporarily restore the liquidity, the `slide` operation will keep scaling down the ANCHOR every time its triggered, as long as `liquidityA > liquidityThreshold`.

## Recommendation

Consider adding a max value to the leverage factor, limiting the reduction of the ANCHOR liquidity. Alternatively, consider using the following implementation for the `getLeverageFactor` function:

```
leverageFactor_ = 1e18 + totalCollateral.mulWad(leverageRange).divWad(_bAssetsCirculating);
```

Where `leverageRange` is a new variable configurable by the protocol. Otherwise consider implementing another implementation taking points 1 & 2 into account.

## Resolution

Baseline Team: The issue was resolved in [PR#65](#).



# M-04 | Liquidity Premium Rounds Down To 0

Category	Severity	Location	Status
Logical Error	● Medium	MarketMaking.sol: 330	Resolved

## Description

The liquidity premium is used to calculate the liquidity of the DISCOVERY range, based on the ANCHOR range liquidity, during rebalances.

The main goal is to scale DISCOVERY range liquidity based on the ratio depending on the tick distance of the active tick to the floor tick, and the TICK\_PREMIUM\_FACTOR.

The issue involves the initialized value of TICK\_PREMIUM\_FACTOR. There is a discrepancy between the tests and the deploy script, as the tests use 4800 but the deploy script uses 4800e18. In case the 4800e18 value is used, it will cause the getLiquidityPremium calculation round to 0, whenever the tick difference is less than 4800. During a sweep and slide this issue will make DISCOVERY liquidity equal to the ANCHOR liquidity.

## Recommendation

Be sure TICK\_PREMIUM\_FACTOR is initialized with the correct value of 4800.

## Resolution

Baseline Team: The issue was resolved in [PR#51](#).

# M-05 | Discovery Liquidity Increased Above MAX\_DISCOVERY\_RATIO

Category	Severity	Location	Status
Logical Error	● Medium	MarketMaking.sol: 227	Resolved

## Description

The ratio between the DISCOVERY and ANCHOR liquidity during a sweep operation should be capped by the MAX\_DISCOVERY\_RATIO (5x), following this logic:

```
liquidityPremium = (liquidityPremium / liquidityA) > MAX_DISCOVERY_RATIO ? liquidityA * MAX_DISCOVERY_RATIO : liquidityPremium;
```

The issue relies on the ratio `liquidityPremium / liquidityA`, as this division rounds down in Solidity. Therefore, every time this division rounds down to the value of MAX\_DISCOVERY\_RATIO (5), then `liquidityPremium` value is not updated, due to the `>` comparison.

Because `liquidityPremium` on its own is larger than `liquidityA * MAX_DISCOVERY_RATIO`, this rounding issue allows the DISCOVERY liquidity to increase up to almost 7x the ANCHOR liquidity during sweep rebalancing.

## Recommendation

Consider updating the comparison to include ratios equal to MAX\_DISCOVERY\_RATIO:

```
liquidityPremium = (liquidityPremium / liquidityA) >= MAX_DISCOVERY_RATIO ? liquidityA * MAX_DISCOVERY_RATIO : liquidityPremium;
```

This prevents `liquidityPremium / liquidityA` ratio to end up being greater than 6x.

## Resolution

Baseline Team: The issue was resolved in [PR#59](#).

# M-06 | Bump Reverts Due To Lack Of Reserves

Category	Severity	Location	Status
DoS	● Medium	MarketMaking.sol: 128	Acknowledged

## Description

An edge case appears when the price is at DISCOVERY range, and all reserves from FLOOR and ANCHOR are borrowed. This could happen when there is heavy buying, and all bAssets are collateralized. If we try to trigger a bump operation, this will revert at this function: `BPOOL.manageLiquidityFor(Range.DISCOVERY, Action.ADD, liquidityD);`

The issue is that when we try to mint liquidity in the Uniswap Pool, the calculation for the token amounts owed to the pool are rounded up: `getAmount1Delta(sqrtRatioAX96, sqrtRatioBX96, uint128(liquidity), true)`.

Due to the rounding up, more reserves are requested to be transferred in the `uniswapV3MintCallback` than available, preventing the bump operation from occurring.

## Recommendation

Prevent the DoS by ensuring that no more reserves are requested for the DISCOVERY range than available:

```
uint256 reservesNeeded = LiquidityAmounts.getAmount1ForLiquidity(
    discovery.sqrtPriceL,
    sqrtPriceA > discovery.sqrtPriceU ? discovery.sqrtPriceU : sqrtPriceA,
    liquidityD
);
uint256 reserveBal = BPOOL.reserve().balanceOf(address(BPOOL));
if (activeTick > discL && liquidityF == 0) BPOOL.manageReservesFor(Range.DISCOVERY,
    Action.ADD, reservesNeeded > reserveBal ? reserveBal : reservesNeeded);
else BPOOL.manageLiquidityFor(Range.DISCOVERY, Action.ADD, liquidityD);
```

## Resolution

Baseline Team: Acknowledged.

# M-07 | Rebalance Threshold May Prevent Sliding Operation

Category	Severity	Location	Status
DoS	● Medium	MarketMaking.sol: 248	Acknowledged

## Description

The REBALANCE\_THRESHOLD is a param set in the constructor of MarketMaking policy. This means that this param can be configured by the protocol.

Although the current deployment script sets the initial value to 1, any value above this will cause unexpected behavior of the slide operation. This is due to the fact that if checkpoint and active ticks are less than 2 TS away from the floor tick, slide can't be triggered if the price trades into the FLOOR range (canSlide tick will be located below the floor tick).

Therefore, the protocol won't be able to correctly rebalance the liquidity when there is heavy selling of bAssets.

## Recommendation

Remove the REBALANCE\_THRESHOLD from the constructor params and set it as a constant inside the MarketMaking policy, with value 1.

## Resolution

Baseline Team: Acknowledged.

# M-08 | Incorrect Circulating Supply Calculation

Category	Severity	Location	Status
Logical Error	● Medium	MarketMaking.sol: 354	Acknowledged

## Description

The `getCirculatingSupply` function removes the current liquidity in `bAssets` from the total token supply. The goal is to determine how many `bAssets` can be sold into the pool.

There are cases when this function returns an outdated value, due to the fact `bAsset` fees are not accounted for in the calculation. Consequently, the value returned might be slightly above the real value.

During heavy selling of `bAssets` into the pool, the difference can increase as there will be more pending fees to claim. This can impact the off-chain systems for managing the operations, as the function will not return the correct state. Even if the system appears solvent using this getter function, a market making operation may revert as the real circulating supply is recalculated during the execution.

## Recommendation

Consider adding the pending fees in `bAssets` to the `getCirculatingSupply` calculation.

## Resolution

Baseline Team: Acknowledged.

# M-09 | Initial Deployed Liquidity Missing Crucial Validations

Category	Severity	Location	Status
Validation	● Medium	InitializeProtocol.sol: 153	Resolved

## Description

For the V2 migration, the InitializeProtocol policy will be used. This will initialize the pool, mint the initial spot supply, setup credits and deploy the pool liquidity.

The issue relies on the deployLiquidity function, as it only verifies the new deployed capacity is above the initial circulating supply. The missing validations are:

- Verify the new liquidityD is not above MAX\_DISCOVERY\_RATIO, as the tick premium and leverage can be high.
- Validate the spot supply is already minted, as the system might be insolvent and distributeSpot does not check this.
- Validate credit is already setup, similar to spot, circulating supply will be minted and can make the system insolvent.

Although there is some documentation about the steps for the V2 migration, the code does not enforce these steps, so there could be arbitrage attacks that can be triggered if the liquidity is not setup correctly.

## Recommendation

Check if the liquidity structure does not allow bump to be executed right away. Additionally, consider adding checks to ensure no more circulating supply is minted after the deployLiquidity is executed.

## Resolution

Baseline Team: Resolved.

# M-10 | Virtual Liquidity Lower Than Expected

Category	Severity	Location	Status
Logical Error	● Medium	InitializeProtocol.sol: 153	Resolved

## Description

When the `slide` operation is triggered, the `virtualLiquidityF` is calculated based on the total collateral and the reserves removed from the FLOOR, using the lower and upper `sqrtPrice` of the entire range.

In case the `slide` is executed when the active tick is in the FLOOR range, `virtualLiquidityF` will still use the upper `sqrtPrice` of the range, instead of the current price. This issue will cause the `virtualLiquidityF` to appear smaller than expected.

## Recommendation

Consider using the correct limit price in the formula:

```
(uint160 sqrtPriceA,,,,,) = BP00L.pool().slot0();
uint256 virtualLiquidityF = uint256(
    LiquidityAmounts.getLiquidityForAmount1(
        floor.sqrtPriceL, sqrtPriceA < floor.sqrtPriceU ? sqrtPriceA : floor.sqrtPriceU,
        CREDIT.totalCreditIssued() + reservesF
    )
);
```

## Resolution

Baseline Team: Resolved.

# M-11 | Operations Using Outdated Circulating Supply

Category	Severity	Location	Status
Logical Error	● Medium	CREDIT.v1.sol: 226	Resolved

## Description

When a credit is defaulted, its collateral is sent directly to the BPOOL contract through the \_burnDefaultedCollateral function. This creates a temporary condition where the circulating supply is higher than expected, if the market making operations are executed before defaulting the expired loans.

In the bump operation, an increased circulating supply will effectively mint more tokens than expected for the inflation basis.

## Recommendation

Ensure defaultOutstanding is executed at the start of market making operations.

## Resolution

Baseline Team: The issue was resolved in commit [f8be1bb](#).



# M-12 | Arbitrage Attack After Slide Operation

Category	Severity	Location	Status
Logical Error	● Medium	MarketMaking.sol: 255	Acknowledged

## Description

The market making slide operation rebalances the liquidity structure when active tick goes below a certain threshold.

In case the heavy selling pushes the price to the FLOOR, the slide operation will be the only range with reserves. When liquidity is added to the FLOOR, bAssets will be minted in this range. This scenario allows arbitrage to buy tokens at discount, trigger sweep/bump operations, and profit from it.

## Recommendation

Verify the initial liquidity structure does not allow these arbitrage attacks to take place.

## Resolution

Baseline Team: Acknowledged.

# M-13 | Attackers Can Borrow With Zero Interest

Category	Severity	Location	Status
Gaming	● Medium	CreditFacility.sol: 126	Resolved

## Description

Users can borrow reserve tokens by adding bAssets as collateral to the system and interests for these credits are paid upfront based on a daily rate and the baseline value.

Credits are created based on the baseline value of the bAssets, and interests are also paid based on the baseline value when the credit is created. However, this baseline value only increases in time, which means that more reserves can be borrowed with the same collateral when the baseline value increases.

And lastly, the protocol [has a check](#) to ensure the borrow is legitimate (either extension of existing borrow or new borrow). But, this check is incorrect and a user can still borrow without extending and without increasing the collateral.

Attackers can combine all of these to borrow zero interest credits:

1. Attacker buys bAssets or already holds.
2. Creates a very long term credit when the blv is as low as possible. The interest is paid at this step.
3. Waits for blv to increase.
4. Borrows again without increasing the collateral and without extending expiry.
5. This second borrow will transfer more reserves due to increased blv, but the interest will be 0.
6. Attacker can do it repetitively every time the blv increases.

The attacker basically setup his future credits, and paid the interest for them at a very low price, making them effectively free.

## Recommendation

Do not allow credits that are not legitimate. Also, charge interest based on the total extra credit instead of charging based on newly added collateral to prevent lost interest in case of blv increase.

## Resolution

Baseline Team: The issue was resolved in [PR#58](#).

# M-14 | liquidityPremium Can Be A Discount

Category	Severity	Location	Status
Logical Error	● Medium	MarketMaking.sol: 212, 296	Resolved

## Description

The liquidity premium calculated in function `getLiquidityPremium()` is calculated as follows:  
`liquidityPremium_ = uint256(uint24(activeTick - BPOOL.floorTick())).divWad(TICK_PREMIUM_FACTOR).`

If the `activeTick` is less than `TICK_PREMIUM_FACTOR` away from the floor tick, then the function returns a proportion less than 100%. Consequently, when the returned proportion is multiplied by the target anchor liquidity in functions `sweep()` and `slide()`, the liquidity amount is decreased rather than increased. This results in unexpected liquidity structures when the active tick is less than 4800 ticks from the floor tick.

For example, the anchor range liquidity is treated as the "top of book" liquidity, and is targeted to have a basis of 1/1000th of the floor liquidity.

However consider a liquidity structure where multiple slides have taken place, and the active tick is 2400 ticks above the floor tick. Now the `liquidityPremium` is  $2400 / 4800 = 0.5$ , resulting in a targeted anchor liquidity of 1/2000th.

The targeted anchor liquidity in this case is less than the configured 1/1000th base ratio, and as the active tick moves closer to the floor tick the targeted anchor liquidity becomes even smaller, which ultimately creates much less price stability when the anchor range is below 4600 ticks in width. Additionally, liquidity is increasingly allocated away from where trading action will accumulate fees when the anchor is collapsing below a width of 4600 ticks.

## Recommendation

When computing the target liquidity threshold for the anchor position, do not allow the anchor position liquidity to drop below the configured 1/1,000th ratio of the floor position liquidity by treating the `liquidityPremium` as a true premium.

```
uint128 liquidityThreshold =
    uint128(virtualLiquidityF
        .mulWad(1e18 + getLiquidityPremium())
        .divWad(ANCHOR_LIQ_THRESHOLD));
```

## Resolution

Baseline Team: Resolved.

# M-15 | getLeverageFactor DoS

Category	Severity	Location	Status
DoS	● Medium	MarketMaking.sol: 339	Resolved

## Description

In the `getLeverageFactor` function the resulting `leverageFactor` is computed by dividing the `totalCollateral` by the `_bAssetsCirculating - totalCollateral`.

However it is possible for the `totalCollateral` to be the entirety of the circulating supply in the event that every holder borrows against their `bAssets`. In this scenario the `getLeverageFactor` will revert with a divide by 0 panic. This results in a DoS for any call to the `sweep` or `slide` functions.

## Recommendation

Add a case to handle the scenario where all circulating assets are being used as collateral in the `getLeverageFactor` function.

If `(_bAssetsCirculating - totalCollateral == 0)` return `maxLeverageFactor`

## Resolution

Baseline Team: The issue was resolved in [PR#65](#).

# M-16 | Risk Free Arbitrage Attack

Category	Severity	Location	Status
DoS	● Medium	MarketMaking.sol	Resolved

## Description

The main market making operations allow the protocol to rebalance the liquidity and bump the floor price, as long as solvency is maintained.

Due to the fact that these operations can be executed in the same transaction, there is an arbitrage attack opportunity that will drain most of the initial reserves from the protocol, without any risk.

Consider the following attack flow:

1. buy bAssets, pushing the price deep into DISCOVERY
2. trigger a sweep to distribute the surplus reserves into the ANCHOR and FLOOR ranges, and update checkpoint tick
3. call bump multiple times to catch up with the active tick.
4. sell all bAssets at a profit, removing a huge chunk of reserves liquidity.

## Recommendation

Limit the amount of times the liquidity operations can be called within a block, or limit it to once every few blocks.

## Resolution

Baseline Team: Resolved.

# L-01 | Slide Prevents Price To Exit The Floor

Category	Severity	Location	Status
Logical Error	● Low	MarketMaking.sol: 295	Acknowledged

## Description

Fees are accumulated in reserves and bAssets in all ranges, and claimed by the market making operations when they are executed. While bAsset fees are burned, the reserve fees are added to either ANCHOR or FLOOR ranges.

Slide operation triggered when the active tick is near the floor tick will claim these reserve fees, but FLOOR is the only range that receives them.

The issue arises when the fees accumulated are significant, compared to the floor reserves. The manageReservesFor function will add liquidity using the balance of the reserves in the BPOOL, and consequently mint more bAssets than expected. This will cause the price to have a hard time exiting the floor, as more reserves will be needed on the way up.

As the floor will end up with higher liquidity than expected, buyers will have less slippage buying bAssets at low prices, opening the possibility of arbitrage attacks, as the capacity will increase due to all the reserves deposited into the FLOOR.

## Recommendation

Consider updating the logic on how bAssets are minted on the FLOOR range when the slide operation is triggered, so the price can exit the range as soon as possible.

## Resolution

Baseline Team: Acknowledged.

# L-02 | Discovery Liquidity Increased After Sliding

Category	Severity	Location	Status
Logical Error	● Low	MarketMaking.sol: 255	Acknowledged

## Description

The `slide` market making operation is triggered when the active tick is below a certain distance from the checkpoint tick. This rebalances the liquidity on the 3 ranges. The `DISCOVERY` liquidity is scaled down during this operation, in favour of increasing the liquidity of lower ranges, increasing the capacity of the system.

After sliding, the `DISCOVERY` range will be readjusted at slightly above the active tick. Therefore, in order to allow price to freely trade up, liquidity in this range should always decrease.

If the system is highly leveraged through heavy borrowing to the point it starts using `ANCHOR` liquidity, then `liquidityA < liquidityThreshold`, and this may cause the `DISCOVERY` liquidity to increase after sliding instead.

## Recommendation

The sweep function should store the liquidity of the discovery range when calling `manageReservesFor()`, and cap the new calculated value based on this amount.

## Resolution

Baseline Team: Acknowledged.

# L-03 | Slide May Result In Equal Anchor And Discovery Liquidity

Category	Severity	Location	Status
Logical Error	● Low	MarketMaking.sol: 300	Acknowledged

## Description

During slide, if liquidityA is equal to liquidityThreshold then both anchor and discovery will be re-deployed with the same liquidity.

This occurs because anchor's liquidity is only reduced by the inverse premium when it is less than liquidityThreshold.

As a result of this edge case, the liquidity structure will deviate from the ideal structure where discovery has more liquidity than anchor.

## Recommendation

Reduce anchor's liquidity by the inverse liquidity premium if liquidityA >= liquidityThreshold. Alternatively, consider increasing discovery's liquidity in this scenario.

## Resolution

Baseline Team: Acknowledged.



# L-04 | Strict Inequality For Anchor Threshold Comparison

Category	Severity	Location	Status
Logical Error	● Low	MarketMaking.sol: 223	Partially Resolved

## Description

In the `sweep` function the `surplusReservesD` are allocated to the anchor and floor positions depending on if it would put the anchor above the desired threshold.

However if the `reservesA + surplusReservesD` would put the anchor position at exactly the desired liquidity threshold, then the surplus is instead split between the anchor in the floor.

In this case it would be more preferable to allocate the surplus to the anchor, as it would put the anchor at exactly the desired liquidity threshold.

## Recommendation

Change the `reservesA + surplusReservesD < thresholdReservesA` comparison to `reservesA + surplusReservesD <= thresholdReservesA`.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-05 | Reserve Token Should Use safeTransfer

Category	Severity	Location	Status
Logical Error	● Low	CreditFacility.sol: 194	Resolved

## Description

The reserve token can be any arbitrary token in future pairs, in order to maintain compatibility with as many tokens as possible, `safeTransfer` and `safeTransferFrom` ought to be used to validate returned values.

## Recommendation

Use `safeTransfer` and `safeTransferFrom` when transferring the reserve token throughout the codebase.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-06 | Inconsistent Default Pattern Array Lengths

Category	Severity	Location	Status
Logical Error	● Low	MarketMaking.sol: 82	Resolved

## Description

In the `configureDependencies` function the dependencies array is initialized with a length of 3, however only 2 entries are written to.

Additionally in the `requestPermissions` function the zeroth entry for the requests array is not assigned to.

## Recommendation

Reduce the size of the dependencies array in the `configureDependencies` function as well as the requests array in the `requestPermissions` function.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-07 | Superfluous brs Address

Category	Severity	Location	Status
Superfluous Code	● Low	BPOOL.sol: 90	Resolved

## Description

In the BPOOL contract there is an address variable named brs which is not referenced nor assigned to.

## Recommendation

Remove the brs address state variable.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-08 | Users May Repay 0 Amount

Category	Severity	Location	Status
Validation	● Low	CreditFacility.sol: 153	Resolved

## Description

In the `repay` function there is nothing preventing a user from repaying with a `_reservesIn` value of 0. This allows for an unexpected action to be taken on the system and will result in an errant `transferFrom` call with a value of 0 in the `updateCreditAccount` function.

Though nothing currently comes of this, the protocol ought to limit unexpected user flows to reduce attack surface.

## Recommendation

Consider validating that the `_reservesIn` value is nonzero in the `repay` function.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-09 | Invalid Event Emission Data

Category	Severity	Location	Status
Logical Error	● Low	CREDIT.sol: 207	Resolved

## Description

In the defaultOutstanding function the Defaulted event is emitted with the wrong order of arguments. The correct order for the Defaulted event is the timeslot, credit, followed by the collateral. However the event emission in the while loop emts with arguments of the timeslot, collateral, followed by the credit.

## Recommendation

Swap the collateral and credit parameters in the Defaulted event.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-10 | Debug Code In Production

Category	Severity	Location	Status
Superfluous Code	● Low	Global	Partially Resolved

## Description

The code includes console2 logs and imports, which are useful for testing, but are completely useless after deployment. These logs will also consume gas when functions are executed.

## Recommendation

Remove any debug imports, logs or TODO comments from the code.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-11 | Misleading Comments

Category	Severity	Location	Status
Documentation	● Low	Global	Resolved

## Description

Some of the comments found in the contracts are either misleading or incorrect, such as:

- BPOOL.V1#L291: The result is based on the checkpointTick instead of the activeTick
- MarketMaking.sol#L152: Incorrect, the activeTick has to be above one tick space from the checkpoint
- MarketMaking.sol#L219: Incorrect, it's 100% of the surplus instead of 90%
- MarketMaking.sol#L44, MarketMaking.sol#L145 and MarketMaking.sol#L164: Old v1 shift function mention.
- MarketMaking.sol#L252: Old v1 bin function mention.

Notice that the lines were taken from the `baseline-team-1-pocs` repo

## Recommendation

Remove or update the previous comments.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).



# L-12 | Unused Parameters

Category	Severity	Location	Status
Superfluous Code	● Low	Global	Partially Resolved

## Description

The contracts include a handful of unused errors, events, variables and imported contracts. These are:

- BPOOL.v1.sol#L71
- BPOOL.v1.sol#L75
- BPOOL.v1.sol#L89
- CreditFacility.sol#L15
- CreditFacility.sol#L17
- CreditFacility.sol#L38
- CreditFacility.sol#L42
- CreditFacility.sol#L47
- CreditFacility.sol#L48
- InitializeProtocol.sol#L41
- MarketMaking.sol#L12
- CREDIT.v1.sol#L226: The event is used but the order of the parameters are incorrect

Notice that the lines were taken from the `baseline-team-1-pocs` repo

## Recommendation

Implement or remove them accordingly.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-13 | Redundant Ternary Operators

Category	Severity	Location	Status
Superfluous Code	● Low	Global	Partially Resolved

## Description

Some of the ternary operators in the contracts are executing redundant operations and should be removed. These include:

- CreditFacility.sol#L141: The prior if condition will prevent an underflow.
- MarketMaking.sol#L200: Code will return early if the current price is lower than the upper tick of the floor.

## Recommendation

Remove them and simply assign the value directly into the variable.

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-14 | Defaulted Event Logging Incorrect Data

Category	Severity	Location	Status
Superfluous Code	● Low	CREDIT.sol	Resolved

## Description

The Defaulted event in the CREDIT module is defined as:  
event Defaulted(uint256 timeslot\_, uint256 credit\_, uint256 collateral\_);

The issue is that when the event is emitted, its using the collateral as the second parameter and credit as the third.

## Recommendation

Update the event to emit the correct data:  
emit Defaulted(timeslotIter, defaultable.credit, defaultable.collateral);

## Resolution

Baseline Team: The issue was resolved in [PR#50](#).

# L-15 | Unnecessary Allowances

Category	Severity	Location	Status
Superfluous Code	● Low	BPOOL.sol: 136	Resolved

## Description

Infinite allowances are given to the Uniswap pool in the `initializePool` function of the `BPOOL` module. Uniswap pool never uses these allowances and they should be removed.

Another point to mention in here is that `msg.sender` of the second `approve` call above is not the `BPOOL` contract but the `InitializeProtocol` contract. Reserve token allowances are given by the `BPOOL` contract itself, but the `bAsset` allowances are given by the `InitializeProtocol` contract.

## Recommendation

Remove unnecessary allowances.

## Resolution

Baseline Team: The issue was resolved in [PR#63](#).

# L-16 | Array Lengths Don't Match During Default Configuration

Category	Severity	Location	Status
Logical Error	● Low	MarketMaking.sol: 82	Resolved

## Description

Protocol uses default framework for smart contracts, which are needed to be configured as modules or policies. Incorrect array sizes are used during the configuration of the MarketMaking contract.

configureDependencies function has a dependency array with a length of 3 while the actual dependency count is 2. requestPermissions function has a permissions array with a length of 8 while the actual permissions count is 7.

## Recommendation

Use the exact array lengths.

## Resolution

Baseline Team: Resolved.

# L-17 | Hardcoded Reserve Token Address Is Incorrect

Category	Severity	Location	Status
Logical Error	● Low	InitializeProtocol.sol: 66	Resolved

## Description

Reserve token address is hardcoded as `address(0x0)` in the `InitializeProtocol` contract. However, in the Blast ecosystem, reserve token address is `0x4300000000000000000000000000000000000004`.

The deployment script uses the correct address. However, the public variable in the contract is incorrectly defined and it is not used.

## Recommendation

Remove the unused variable from the contract.

## Resolution

Baseline Team: Resolved.

# L-18 | totalCreditIssued Includes Interest Which Has Yet To Be Paid

Category	Severity	Location	Status
Logical Error	● Low	CreditFacility.sol:138	Resolved

## Description

When a user borrows, both principal (loan amount) and interest are added to the user's account and to the global variable `totalCreditIssued`. However, interest is only to be paid after `repay`, so it should not be treated as credit at time of borrow.

By including interest in `totalCreditIssued`, `calculateTotalCapacity` is optimistically inflated before the interest is actually paid and added as liquidity. For accuracy and a more conservative approach, interest should be excluded from `totalCreditIssued`.

However, there were no risks identified in including the interest since the collateral for the loan is held by the protocol (i.e., it cannot be sold). Even if the loan defaults, burning the loan's collateral improves overall solvency.

## Recommendation

Clearly document this behavior to users.

## Resolution

Baseline Team: Resolved.

# L-19 | Floor Tick Increment Prevents Sliding Operations

Category	Severity	Location	Status
Validation	● Low	BPOOL.v1.sol	Resolved

## Description

The bump operation in MarketMaking contract increments the floorTick as long as the new floor does not surpass the active and checkpoint ticks. This operation will revert if:

- $activeTick < floorTick + (TICK\_SPACING * 2)$
- $checkpointTick < floorTick + (TICK\_SPACING * 2)$

The issue is that these validations allow the floorTick to end up exactly one TS below the checkpoint and activeTick. Consider this scenario:

1. floor tick is at 200
2. Price trades higher, ends up in tick 600 exactly.
3. sweep rebalances liquidity and sets checkpoint to 600
4. bump is executed and increments floor tick to 400 (still below active and checkpoint)

This scenario will prevent slide operations to be executed even if price drops into FLOOR range, as the active tick will need to go below the floor tick to pass the validation:

$$activeTick < (BPOOL.checkpointTick() - (TS * int24(int8(REBALANCE\_THRESHOLD))))$$

## Recommendation

Consider updating the incrementFloorTick validations, and use `<=` instead of `<` for the comparison against the floorTick value.

## Resolution

Baseline Team: Resolved.



# 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>