GA GUARDIAN

Baseline

Loops Updates

Security Assessment

February 25th, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Daniel Gelfand, Osman Ozdemir

Client Firm Baseline

Final Report Date February 25, 2025

Audit Summary

Baseline engaged Guardian to review the security of their Loops Updates. From the 17th of August to the 23rd of August, a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 6 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: **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-Perps

Table of Contents

Project Overview

Project Summary

Project Name	Baseline
Language	Solidity
Codebase	https://github.com/0xBaseline/baseline-v2
Commit(s)	Initial commit: 102c7a3abcd34b3c1f97efd400e3a7af4afbaf6b Final commit: a20a6625f58e1e54f06ca92d2a4cd5f4d6c40c64

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	0	0	0	0	0	0
High	6	0	0	0	0	6
Medium	4	0	0	1	0	3
Low	10	0	0	1	0	9

Audit Scope & Methodology

Vulnerability Classifications

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

Impact

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

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

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

Likelihood

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

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

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

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

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

The auditing process pays special attention to the following considerations:

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

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Dangerous setFundingRate Function	Logical Error	High	Resolved
H-02	closePosition May Break Through The Floor	Logical Error	High	Resolved
H-03	Crucial Storage Overwritten On configureDependencies	Access Control	High	Resolved
H-04	Loops Vault Decay Invalidates Solvency Check	DoS	High	Resolved
H-05	Trapped Fees In LoopFacility	Logical Error	High	Resolved
H-06	Missing Blast Configurations	Best Practices	High	Resolved
M-01	MarketMaking Ignores Loops Capacity	Logical Error	Medium	Resolved
M-03	Vault Position Not Sum Of User Positions	DoS	Medium	Resolved
M-04	Slide Causes Anchor To Disappear	Warning	Medium	Acknowledged
M-05	Incorrect Virtual Reserves Accounting	Validation	Medium	Resolved
L-01	Unnecessary tradingInFloor Case	Optimization	Low	Resolved
L-02	Lacking Use Of Tick Spacing Constant	Best Practices	Low	Resolved
L-03	Anchor Can Exceed Defined Width	Documentation	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-04	Anchor Ticks Crossed In Drop	DoS	Low	Resolved
L-05	Position Can Increase Without Debt	Logical Error	Low	Resolved
L-06	Typo	Typo	Low	Resolved
L-07	Zero Transfer DoS	DoS	Low	Resolved
L-08	Unexpected Deployment Behavior	Configuration	Low	Resolved
L-09	Infinite Slide Glitch	Warning	Low	Resolved
L-10	Unsafe Casting	Casting	Low	Resolved

H-01 | Dangerous setFundingRate Function

Category	Severity	Location	Status
Logical Error	High	Loops.v1.sol: 190	Resolved

Description

The setFundingRate function allows a trusted address to assign the funding rate for the vault, however in almost any circumstance where the fundingRate is updated it will invalidate the funding accounting of the vault.

For example:
• fundingRate is 10% per year
• Position A is opened at year 0 with X collateral
• Position B is opened at year 1, with X collateral
• The vault experiences decay for 1 year, collateral: X * 1/e^0.1 = X * 1/1.105 and then gains X collateral
• Vault collateral is now X * (1 + 1/1.105) = 1.905X
• The funding rate is set to 5% per year
• Position A is closed at year 2, the position experiences decay for 2 years at a rate of 5% per year, collateral: X * 1/e^0.1 = X * 1/1.105 = 0.905X
• The vault experiences decay for 1 year at 5%, collateral: 1.905X * 1/e^0.05 = 1.905X * 1/1.051 = 1.813X
• Position A's collateral is now removed from the vault, collateral left is 1.813X - 0.905X = 0.908
• Position B is closed at year 2, the position experiences decay for 1 year at a rate of 5%, collateral: X * 1/e^0.05 = X * 1/1.051 = 0.951X
• Position B's collateral is attempted to be removed from the vault, but the vault only has 0.908X collateral left so the position cannot be fully closed!

The core issue is that every position must be updated to agree with the decay experienced by the vault, if the rate changes then every position must be updated along with the vault to decay at that time with the previous rate.

Recommendation

Updating every position to update the rate is not feasible in an EVM environment, consider restructuring the decay to rely on a single fundingDecayAcc which accumulates for every position in the vault, similar to a rewardsPerShare model.

Every position can be stamped with a lastFundingDecayAcc and the decay can be measured as the difference between the latestFundingDecayAcc() - position.lastFundingDecayAcc. Then the setFundingRate function can simply update the lastFundingDecayAcc using the previous rate before assigning the new rate.

Resolution

Baseline Team: The issue was resolved in commit.

H-02 | closePosition May Break Through The Floor

Category	Severity	Location	Status
Logical Error	High	LoopFacility.sol: 159	Resolved

Description

The `closePosition` function swaps bAssets for reserves before adding reserves back to the floor position, therefore it is possible for the bAsset sell to break through the floor tick and invalidate BLV.

Recommendation

At the end of `closePosition` revert if `_tradingInFloor` is true, similar to in `_deleverage`.

Resolution

Baseline Team: The issue was resolved in commit.

H-03 | Crucial Storage Overwritten On configureDependencies

Category	Severity	Location	Status
Access Control	High	MarketMaking.sol: 154-157	Resolved

Description

In the `configureDependencies` function the `sweepTick`, `slideTick`, and `lastDropTimestamp` are assigned to their initial values. However there is no access control that prevents the `configureDependencies` function from being called again.

Recommendation

Consider only assigning these values on the first call to `configureDependencies`, and if necessary include a separate trusted function to re-assign the values.

Resolution

Baseline Team: The issue was resolved in commit.

H-04 | Loops Vault Decay Invalidates Solvency Check

Category	Severity	Location	Status
DoS	High	Global	Resolved

Description PoC

The Loops vault decays the debt of every loop position over time and this decay is reported by the `totalDebt`, however the `totalDebt` function does not reduce the total circulating supply corresponding to the decay of the total position collaterals.

As a result the decay will invalidate the solvency check and DoS all functionality until funding has been charged.

Recommendation

Charge funding before completing every action in the system that relies on the solvency check, or consider accounting for the circulating supply that would decrease from the decay in the solvency check.

Resolution

Baseline Team: The issue was resolved in commit.

H-05 | Trapped Fees In LoopFacility

Category	Severity	Location	Status
Logical Error	High	LoopFacility.sol	Resolved

Description

In the loop facility fees are often collected to the contract with the `_pullReserves` function, however there is no functionality to retrieve these fees.

Recommendation

Implement a function similar to the `setFeeRecipient` function in the `CreditFacility` to retrieve these fees.

Resolution

Baseline Team: The issue was resolved in commit.

H-06 | Missing Blast Configurations

Category	Severity	Location	Status
Best Practices	High	LOOPS.v1.sol: 50	Resolved

Description

In the `LOOPSv1` module there is no configuration for blast yields in the constructor.

Recommendation

Add blast yields configuration to the constructor.

Resolution

Baseline Team: The issue was resolved in commit.

M-01 | MarketMaking Ignores Loops Capacity

Category	Severity	Location	Status
Logical Error	Medium	MarketMaking.sol: 553	Resolved

Description

Additional debt can now serve as capacity for the Baseline system in the Loops vault. This is accounted for in the `CreditFacility` but not in the `MarketMaking` contract. As a result the capacity checks between the `CreditFacility` and `MarketMaking` policies will not agree.

Recommendation

Include the `LOOPS.totalDebt()` when computing the capacity in the `MarketMaking` contract.

Resolution

Baseline Team: The issue was resolved in commit.

M-03 | Vault Position Not Sum Of User Positions

Category	Severity	Location	Status
DoS	Medium	LOOPS.v1.sol	Resolved

Description

Function `getFundingSince` is not perfectly precise, such that the decay of two time deltas X and Y is not the same as the funding decay of one time delta X + Y. This is important since `chargeFunding` only updates the last update timestamp for the vault position, not user positions.

Consider this scenario where there is only one open position:
1) 10 seconds pass.
2) Vault is charged funding for 10 second decay.
3) 200 more seconds pass.
4) Vault is charged funding for 200 second decay; User is charged funding for 210 second delay.
5) User sends request to close their position.

Ultimately, the latest position of the vault is not aligned with the latest position of the user due to the imprecision of `getFundingSince`. The position the user can reduce is greater than the latest vault position, causing an underflow when performing `vault.position = _positionToReduce`.

This can be harmful in the case there are multiple open positions, and a single depositor is left hanging and unable to close their position. Note that this issue is also applicable to the `vault.debt = debtToReduce_;` calculation as the debt is also updated when funding is charged.

Recommendation

Change the reduction to:

```
uint256 amtPosToReduce = vault.position < _positionToReduce * vault.position:
_positionToReduce;

vault.position = amtPosToReduce; uint amtDebtToReduce = vault.debt < debtToReduce_ *
vault.debt : debtToReduce_; vault.debt = amtDebtToReduce; // bAsset.transfer(msg.sender,
amtPosToReduce);
```

Resolution

Baseline Team: Resolved.

M-04 | Slide Causes Anchor To Disappear

Category	Severity	Location	Status
Warning	Medium	MarketMaking.sol: 410	Acknowledged

Description

In the slide function the reserves of the anchor position are added back to the anchor after potentially extending the Anchor further downwards with a call to `_updateTicks`.

This can result in the Anchor position having little liquidity or disappearing entirely after the slide operation. This is because the price may have been set less than or equal to the lower end of the Anchor position, in which case there would be no reserves in the liquidity position.

In this case the Anchor position would not be built up again until a sweep occurs, and in the meantime there can be erratic price fluctuations between the floor and discovery which may be far apart.

Recommendation

In these situations consider keeping the Anchor position to the `tickSpacing` above the current price so that the Anchor does not entirely disappear, but is not extended below the active price because that would require reserves to be pulled from the floor.

Otherwise be aware of this quirk in the system and document it for users and integrators.

Resolution

Baseline Team: Acknowledged.

M-05 | Incorrect Virtual Reserves Accounting

Category	Severity	Location	Status
Validation	Medium	BaselineInit.sol: 214	Resolved

Description

In the launch function the `pessimisticCapacity` includes the floor reserves when stretching the virtual reserves over the entire floor position to compute its worst case capacity.

This incorrectly accounts for stretching out the floor reserves which would actually increase if the price were to rise to the upper floor tick.

This was the original reason why the virtual reserves had to be stretched — because they would not receive corresponding reserves in as price rose to the upper tick of the floor.

This accounting is attempted to be fixed by leaving the `bAssets` of the floor position in the circulating supply, as if they had been swapped out of the floor as price rose. However this again does not account for the reserves of the floor increasing due to swap input amounts.

Recommendation

Remove the special accounting for the floor position and revert to the original solvency check that was previously present in the launch function, with the one addition of the `LOOPS.totalDebt()` value in the `pessimisticCapacity` accounting.

Resolution

Baseline Team: The issue was resolved in commit.

L-01 | Unnecessary tradingInFloor Case

Category	Severity	Location	Status
Optimization	Low	LoopFacility.sol: 252, 258	Resolved

Description

Since the `floor.bAssets` will only be nonzero if the price is inside of the floor, the `floor.bAssets` can just be removed from the subtraction of `BPOOL.totalSupply` instead of using the special `_tradingInFloor` case.

Recommendation

Remove the special case handling and remove the `floor.bAssets` from the subtraction of `BPOOL.totalSupply`.

Resolution

Baseline Team: Resolved.

L-02 | Lacking Use Of Tick Spacing Constant

Category	Severity	Location	Status
Best Practices	Low	MarketMaking.sol: 545	Resolved

Description

In the `getCurrentThreshold` the full tick spacing below the `sweepTick` is computed, however the computation uses a direct subtraction of 200 rather than the `T_S` constant which is determined by the BPOOL.

Recommendation

Use the `T_S` rather than a hardcoded spacing of 200.

Resolution

Baseline Team: The issue was resolved in commit.

L-03 | Anchor Can Exceed Defined Width

Category	Severity	Location	Status
Documentation	Low		Acknowledged

Description

The slide function will now no longer move the `anchorUpper/discoveryLower` tick down, and instead only extend the lower tick of the anchor range downwards.

This is because the `_updateTicks` function will not update the `sweepTick` but will set the lower anchor tick as the anchor width below the `activeTS` which has indeed changed.

Recommendation

This may be expected behavior, if so then consider documenting clearly that the Anchor can exceed the defined width.

Resolution

Baseline Team: Acknowledged.

L-04 | Anchor Ticks Crossed In Drop

Category	Severity	Location	Status
DoS	Low	MarketMaking.sol: 481	Resolved

Description

In the drop function when the tick range is assigned to the Anchor position in `_decrementSweepTick` it is possible for the `targetSweepTick` to be lower than the `anchorTickL` in rare cases where there is a large gap between the Anchor and Floor ranges which price has traversed.

This will result in an `InvalidTickRange` revert and disallow the drop from occurring. The workaround is to simply call slide before dropping so that the anchor can sufficiently extend downwards.

Recommendation

Consider adding this case to the `return false` case in `_decrementSweepTick` to explicitly revert with the `CannotDropDiscovery` error or consider adding a revert specific to cases where `slide` must be called first.

Resolution

Baseline Team: The issue was resolved in commit.

L-05 | Position Can Increase Without Debt

Category	Severity	Location	Status
Logical Error	Low	LoopFacility.sol: 124	Resolved

Description

If `_totalCollateral` is a very small value such as 20 wei, `debt_ = _totalCollateral.mulWad(BPOOL.getBaselineValue());` will output 0 due to precision loss, and the position will increase without a corresponding increase in debt.

Recommendation

Consider validating that the debt is non-zero when opening a position.

Resolution

Baseline Team: The issue was resolved in commit.

L-06 | Typo

Category	Severity	Location	Status
Typo	Low	LoopFacility.sol: 118	Resolved

Description

The documentation for function `openPosition` contains the typo `posisble` which should be updated to `possible`.

Recommendation

Update the typo.

Resolution

Baseline Team: The issue was resolved in commit.

L-07 | Zero Transfer DoS

Category	Severity	Location	Status
DoS	Low	LoopFacility.sol: 151, 190	Resolved

Description

In the `openPosition` function a refund is issued to the user when the maximum reserves are not fully used, however even when the `reservesIn_ = _maxReservesIn` the transfer is still initiated.

For some reserve tokens this may cause a revert due to the zero amount transfer. Additionally another potential zero transfer occurs in the `closePosition` function on line 190.

Recommendation

Add an if case to both of these transfers so that they are only attempted if there is a nonzero transfer amount.

Resolution

Baseline Team: The issue was resolved in commit.

L-08 | Unexpected Deployment Behavior

Category	Severity	Location	Status
Configuration	Low	Global	Resolved

Description

When deploying the new `MarketMaking` policy the `slideTick` will be assigned to the current active tick spacing. However this tick spacing may be in the current Discovery range, which can lead to a minor unexpected state.

Recommendation

Be sure to deploy the new `MarketMaking` system when the active price is within the Anchor position.

Resolution

Baseline Team: Resolved.

L-09 | Infinite Slide Glitch

Category	Severity	Location	Status
Warning	Low	MarketMaking.sol	Resolved

Description

When the `activeTick` is on an even tick spacing and the `slideTick` is the direct tick spacing above then a user can invoke a slide over and over again.

This is because the criteria for a slide is:
`activeTick = slideTick - TS`

And `_updateTicks` performs:
`slideTick = activeTS`

Recommendation

Consider requiring `activeTick < slideTick - TS` to trigger a slide.

Resolution

Baseline Team: Resolved.

L-10 | Unsafe Casting

Category	Severity	Location	Status
Casting	Low	MarketMaking.sol: 465	Resolved

Description

Inside `_decrementSweepTick` the following calculation is performed: `uint256 discoveryPremiumTS = uint256(uint24((sweepTick - activeTS) / T_S));` The issue is that the `uint24` may potentially cast a negative value, causing silent overflow.

For Example:

`sweepTick` = -64400
`activeTS` = -64200
Difference = -200
`(sweepTick - activeTS) / T_S)` = -1
`uint24((sweepTick - activeTS) / T_S)` = `uint24(-1)` = 16777215

`discoveryPremiumTS` becomes much larger than it should be, which leads to an overflow when performing `(tickSpacingsToDecay * T_S))`. Consequently, dropping liquidity is prevented from occurring due to panic overflow.

Recommendation

Consider using `SafeCast`.

Resolution

Baseline Team: The issue was resolved in commit.

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