

Morpho: Midnight Competition Fixes Security Review

Auditors

MiloTruck, Lead Security Researcher

Saw-mon and Natalie, Lead Security Researcher

StErMi, Lead Security Researcher

Jonatas Martins, Security Researcher

Report prepared by: Lucas Goiriz

August 10, 2026

Contents

1 About Spearbit **2**

2 Introduction **2**

3 Risk classification **2**
3.1 Impact . . . 2
3.2 Likelihood . . . 2
3.3 Action required for severity levels . . . 2

4 Executive Summary **3**

5 Findings **4**
5.1 Low Risk . . . 4
5.1.1 Foreign ratifiers can import delegated signer authority from another Midnight instance . . . 4
5.1.2 Allow for a modular time to max lif ramping . . . 4
5.2 Gas Optimization . . . 5
5.2.1 Caching `_position.debt` can be hoisted up . . . 5
5.3 Informational . . . 5
5.3.1 The new seller-state checks can block inner-take trees that may help protocol and lender solvency in edge cases . . . 5
5.3.2 `EventsLib.Take` emits a coarser parameter regarding some accounting changes . . . 7

6 Appendix **9**
6.1 Subset of competition findings . . . 9
6.1.1 Description . . . 9
6.2 Recovery close factor check behavioural changes . . . 10
6.2.1 Description . . . 10
6.3 Added parameters to `Market` structure . . . 10
6.3.1 Description . . . 11
6.3.2 Recommendation . . . 11
6.3.3 Morpho . . . 11
6.3.4 Footnote . . . 11
6.4 Morpho Competition Finding 1086 Analysis . . . 11
6.4.1 Description . . . 11
6.4.2 Footnote . . . 13
6.4.3 Notations . . . 15

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Morpho is a trustless and efficient lending primitive with permissionless market creation.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Morpho: Midnight Competition Fixes according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 1 days in total, Morpho engaged with Spearbit to review the midnight protocol. In this period of time a total of **5** issues were found.

Summary

Project Name	Morpho
Repository	midnight
Commit	3836155f
Type of Project	DeFi, AMM
Audit Timeline	Jun 29th to Jun 30th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	2	0	2
Gas Optimizations	1	0	1
Informational	2	0	2
Total	**5**	**0**	**5**

The Spearbit team reviewed Morpho's midnight holistically on commit hash e6f2bf28 and concluded that all findings were addressed and no new vulnerabilities were identified.

5 Findings

5.1 Low Risk

5.1.1 Foreign ratifiers can import delegated signer authority from another Midnight instance

Severity: Low Risk.

Context: EcrecoverRatifier.sol#L34, SetterRatifier.sol#L31.

Description:

> **i Note:**
> This is a regression bug that was re-introduced in PR 964 after being fixed in PR 744. The original finding was documented in here.

`Midnight.take` only checks that the maker authorized `offer.ratifier` in the current Midnight instance before delegating trust to the external ratifier. `EcrecoverRatifier.onRatify`, however, validates the recovered signer against its own immutable `MIDNIGHT` address instead of the caller.

As a result, if a maker authorizes the same ratifier R on multiple Midnight deployments M_A, M_B, a signer who is authorized for that maker on M_A can ratify an order that executes on M_B. This crosses an otherwise natural trust boundary: delegated signer permissions are expected to be local to each Midnight deployment, but the ratifier imports them from a different instance.

The new changes add the `midnight` address to the `market` structure that would pin it to the `Midnight` contract it gets touched into.

Recommendation: The newly introduced changes add the `midnight` address to the `market` structure that would pin it to the `Midnight` contract it gets touched into. So unlike before, when one had the check `require(msg.sender == MIDNIGHT)`, one can instead perform the following check:

```
require(offer.market.midnight == MIDNIGHT, NotMidnight());
```

This way two different official deployment of `Midnight` cannot by mistake call into the same `EcrecoverRatifier` and reuse the authorization set of the ratifier's `MIDNIGHT` address. While 3rd party entities can still call into this endpoint to check whether the offer is ratified.

It could also be useful to potentially add an extra check:

```
require(offer.ratifier == address(this), InvalidRatifier());
```

which would prevent 3rd party offer ratification foot-guns.

The above checks should also be added to `SetterRatifier` contract since it has a similar issue. Moreover, make sure the above scenario is implemented fully in the test suite to avoid codebase regression in the future.

Morpho: Acknowledged.

Spearbit: Acknowledged.

5.1.2 Allow for a modular time to max lif ramping

Severity: Low Risk.

Context: ConstantsLib.sol#L19.

Description: The constant `TIME_TO_MAX_LIF` encodes a risk factor for both lenders and borrowers. This value has been changed from `15 minutes` to `60 minutes`. It would make sense to allow for a more modular parameter that can be baked into the `Market` thus allowing all the parties to pick their desired risk parameters attached to the `Market`.

Recommendation: Instead of hardcoding the `TIME_TO_MAX_LIF` introduce a more modular design where this value can be added as a parameter in the `Market` structure. Then perform a restrictive set check upon 1st touch of the market using:

- Upper and lower bound checks using fixed constant min and max values or.
- Configurator allowed list of values.

Morpho: Acknowledged.

Spearbit: Acknowledged.

5.2 Gas Optimization

5.2.1 Caching `_position.debt` can be hoisted up

Severity: Gas Optimization.

Context: Midnight.sol#L639.

Description/Recommendation: Caching `_position.debt` can be hoisted up before the following check:

```
require(_position.debt > 0, NotBorrower());
```

which would allow one to replace the check with:

```
uint256 originalDebt = _position.debt;
require(originalDebt > 0, NotBorrower());
```

Morpho: Acknowledged.

Spearbit: Acknowledged.

5.3 Informational

5.3.1 The new seller-state checks can block inner-take trees that may help protocol and lender solvency in edge cases

Severity: Informational.

Context: Midnight.sol#L511.

Description: The `seller` state transition checks in `take` used to be:

```
require(
       position[id][seller].debt == 0
    || liquidationLocked(id, seller)
    || (
            block.timestamp <= offer.market.maturity
         && isHealthy(offer.market, id, seller)
       ),
    SellerIsLiquidatable()
);
```

The current implementation splits this check into:

```
require(block.timestamp <= offer.market.maturity || sellerDebtIncrease == 0, ...);

// other checks, state updates, callbacks, token transfers, ...

require(liquidationLocked(id, seller) || isHealthy(offer.market, id, seller),
↪  SellerIsLiquidatable());
```

1. pre-maturity: `block.timestamp <= offer.market.maturity` is true. Thus.

1.1 previous checks: The checks reduce to:

```
.
   position[id][seller].debt == 0
|| liquidationLocked(id, seller)
|| isHealthy(offer.market, id, seller)
```

If the seller position is not healthy then that would imply that `position[id][seller].debt` is not `0`. Thus the above check can also be reduced to:

```
.
   liquidationLocked(id, seller)
|| isHealthy(offer.market, id, seller)
```

1.2 current checks: The 1st check passes pre-maturity so not included below:

```
.
   liquidationLocked(id, seller)
|| isHealthy(offer.market, id, seller)
```

1.3 pre-maturity summary: Previous and current checks are equivalent.

2. post-maturity: `block.timestamp <= offer.market.maturity` is false. Thus.

2.1 previous checks: The checks reduce to:

```
.
   position[id][seller].debt == 0
|| liquidationLocked(id, seller)
```

2.2 current checks: The 1st check passes pre-maturity so not included below:

```
require(sellerDebtIncrease == 0, ...);

// other checks, state updates, callbacks, token transfers, ...

require(liquidationLocked(id, seller) || isHealthy(offer.market, id, seller),
↪  SellerIsLiquidatable());
```

2.3 post-maturity summary: 2.3.1 seller's liquidation **IS** locked

- **before**: checks pass.

- **now**: checks only pass if `sellerDebtIncrease` is `0`. So post-maturity when the liquidation is locked (we are in the inner call frame into `take` from an outer `take` call frame), the seller cannot increase its debt.

The new checks are more **strict**.

2.3.2 seller's liquidation IS NOT locked:

- **before**: checks only pass if the seller does not have any debt.
- **now**: checks reduce to `sellerDebtIncrease == 0` being true (check block 1) and `isHealthy(offer.market, id, seller)` being true (check block 2). Thus the checks only pass if the seller does not increase its debt and keep its position healthy. Basically repaying debts with potentially better prices and maybe purchasing some credits.

So in this case the new checks are **looser** more in favour of the overall protocol/lender solvency.

Recommendation: **2.3.1** needs to further be documented and perhaps analyzed as it would restrict some potential edge use cases that would help with protocol/lender solvency.

Morpho: We checked that it wouldn't restrict any fair use-case. one thing that we were worried about it crossed offers (buy higher than sell) arbitrages. but the arbitrager can start by buying and then sell, they don't need to increase their debt.

About documenting the behaviour, we think that it's quite simple and self-documenting now.

Footnote: There is also a comment about this change in PR 940 which is not completely accurate regarding the pre-maturity phase.

Morpho: Acknowledged.

Cantina Managed: Acknowledged.

5.3.2 `EventsLib.Take` emits a coarser parameter regarding some accounting changes

Severity: Informational.

Context: Midnight.sol#L457-L477.

Description: Changes made in PR 973:

```
  emit EventsLib.Take(
      msg.sender,
+     keccak256(abi.encode(offer)),
      id,
-     units,
-     taker,
-     offer.maker,
      offer.buy,
+     offer.maker,
      offer.group,
+     offer.ratifier,
+     ratifierData,
+     units,
+     taker,
      buyerAssets,
      sellerAssets,
      newConsumed,
      buyerPendingFeeIncrease,
```

```
      sellerPendingFeeDecrease,
-     buyerCreditIncrease,
-     sellerCreditDecrease,
+     // forge-lint: disable-next-line(unsafe-typecast)
+     int256(buyerCreditIncrease) - int256(sellerCreditDecrease),
      receiver,
      payer
  );
```

regarding `buyerCreditIncrease` and `sellerCreditDecrease` the previous version was more fine-grained.

Recommendation: Document the decision as to why an aggregated coarser value is emitted.

Morpho: Acknowledged.

Spearbit: Acknowledged.

6 Appendix

6.1 Subset of competition findings

Severity: Informational

Context: IMidnight.sol#L38, Midnight.sol#L54-L55, Midnight.sol#L377

6.1.1 Description

- **2446**: interesting griefing vector. But agree with **Adrien LF** regarding liquidators potentially routing their calls through another contract.
- **1086**: See Finding 9
- **1074**: These scenarios have been considered during previous reviews, (dust borrow positions and whether min amounts can be implemented). One thing is that the group of sybil adversaries would need to lock a proportionally higher collateral (depending on `lltv`$_j$). One should also note that splitting a take into multiple takes for a taker would require more collateral (due to roundings) plus there could also be trading fees..

  > ... If collateral price falls during that delay

  Then in this case the attack can actually become profitable while also liquidations are also not incentivized.

  > **i Note:**
  > also that these limits could also be implemented in the seller and buyer callbacks since `units` is provided to them.

- **636**: This is withdrawn. But we also had filed a similar finding. The finding is also badly formatted..
- **461**: Also can be avoided if the liquidation was routed through a contract.
- **49**: Depends on some nuances on how pending fees should be continuously taken from credit holders. The finding discussed allows lazily to update the future un-collected fees using the new `_marketState.continuousFee` value (the entity behind seller and buyer would need to pay trading fee for this update).

  NatSpec has been added regarding this finding:

  > If the market's continuous fee is decreased lenders might self-take to exit and re-enter to reduce their pending fee (at the cost of the settlement fee).

- **584**: Based on previous discussions of researchers with the Morpho team, the protocol is designed with an intent to discourage makers gating the taker sets. That is why the taker is not passed on to the ratifier and also that is why counter-parties are not passed to the buyer or seller callbacks. The gating mechanism is delegated to the `enterGate` address in the `take` route.
- **498**: This has been fixed by introducing new slippage protection parameter `continuousFeeCap` in the `Offer` struct.

  > **i Note:**
  > The check below could have also been enforced in an `IRatifier` implementation.
  >
  > ```
  > require(_marketState.continuousFee <= offer.continuousFeeCap,
  > ↪  ContinuousFeeAboveOfferCap());
  > ```

Morpho: Acknowledged.

Cantina Managed: Acknowledged.

6.2 Recovery close factor check behavioural changes

Severity: Informational

Context: Midnight.sol#L692-L703

6.2.1 Description

The checks in this context used to be:

```
if (block.timestamp <= market.maturity) {
   ...
   uint256 maxRepaid = lltv < WAD
      ? ...
      : type(uint256).max;

   // RCF check
   require(repaidUnits <= maxRepaid || ... < market.rcfThreshold, ...);
}
```

when `lltv == WAD`, `maxRepaid` would have ended up being `type(uint256).max`. Thus the 1st condition would automatically be satisfied. And since the definition of `lltv` in the current implementation has been hoisted out of the `if` block one can completely skip this block by introducing the new 2nd conditional statement for the `if` block. ie:

```
if (block.timestamp <= market.maturity && lltv < WAD) { ... }
```

But the actual current conditional is:

```
if (!postMaturityMode && lltv < WAD) { ... }
```

That would mean in the new implementation, the `RCF` check can be extended post-maturity as well if one would like to avoid the `TIME_TO_MAX_LIF` ramp to `_maxLif`:

- `postMaturityMode == false` and the position is unhealthy.

So:

- **previous behaviour**: `RCF` check could only be applied to unhealthy positions pre-maturity
- **current behaviour**: `RCF` check can be applied to unhealthy positions pre or post maturity. To avoid the check post-maturity the liquidator would need to set `postMaturityMode` to `true` which could cause the `lif` to use the ramp value (not the max lif initially).

Morpho: Acknowledged.

Cantina Managed: Acknowledged.

6.3 Added parameters to `Market` structure

Severity: Informational

Context: IMidnight.sol#L5-L14

6.3.1 Description

```
  struct Market {
+    uint256 chainId;
+    address midnight;
     address loanToken;
     ...
  }
```

These new parameters added would pin:

- the `midnight` deployment instance and...
- `chainid`.

This is mostly important in the context of `SetterRatifier` to distinguish between the offer trees across different deployments of `Midnight`. With the new change proof of offer inclusion in the tree across deployments cannot be reused.

In other words, with the new implementation one can have offer leaf nodes corresponding to different deployments. But each offer lead node can only be used with one specific deployment (ie, the leaf nodes are now **pinned** to specific deployments, not considering forks). Whereas before the leaf nodes were not pinned to deployments and could have been reused across different ones.

6.3.2 Recommendation

It would be best to document the decision that led to major changes such as above. Perhaps a `CHANGELOG` file providing more context for major changes would be beneficial when cutting new releases.

6.3.3 Morpho

The main motivation is to be able to easily distinguish offer's chains in a future where we have an offchain mempool common to all chains. but it has other benefits, such as Market structs being unique identifiers (instead of only the id), and the fact that we don't need anymore the `INITIAL_CHAIN_ID` variable

6.3.4 Footnote

See PR 1016 comment.

6.4 Morpho Competition Finding 1086 Analysis

Severity: Informational

Context: *(No context files were provided by the reviewer)*

6.4.1 Description

In this analysis the main concern is not issues related to roundings. Thus we omit rounding below for easier analysis.

The investigation below is regarding scenarios stemmed from Morpho Midnight competition finding **1086**.

We also might omit some of the sub/super script since the context, the borrower, the market/obligation is fixed. So:

$$
\begin{aligned}
D &= D_{\mathcal{O},u} \\
D_m &= D^{max}_{\mathcal{O},u} \\
D_d &= D^{dis}_{\mathcal{O},u}(\texttt{lif}^{max}_{\mathcal{O},_}) \\
a_r &= a_{repaid}
\end{aligned}
$$

$$a_m^j = a_{repaid}^{max,j}$$

> **i Note:**
> We assume the position has no bad debt at the pre-liquidation price vector, i.e. $D \leq D_d$. Otherwise, `liquidate` first realizes $D - D_d$ as bad debt before applying the repayment/seizure step, so the state transition analyzed below would need an additional initial debt reduction.

Let

$$\alpha_j = \frac{\texttt{lif}_{\mathcal{O},T_j}^{max}}{10^{18}} \in [1, 2]$$

$$\beta_j = \frac{\texttt{lltv}_{\mathcal{O},T_j}}{10^{18}} \in [0, 1]$$

$$\gamma_j = \alpha_j \cdot \beta_j \in [0, 0.999]$$

where j is the collateral index. There is also the special case where β_j could be 1 which would force α_j to be equal to 1 and thus $\gamma_j = 1$

Without loss of generality one can assume γ_j are ordered:

$$\gamma_0 \leq \gamma_1 \leq \cdots \leq \gamma_n$$

If we had:

$$\gamma_i \leq \frac{D_m}{D} < \gamma_{i+1}$$

we would like to analyse the effect of liquidation (partial or max) using $j \in \{i+1, \cdots, n\}$. We have:

$$a_r \leq \frac{c_j \cdot p_j \cdot \beta_j}{\gamma_j} \leq \frac{D_m}{\gamma_j} < D < \frac{D - D_m}{1 - \gamma_j} \approx a_m^j$$

> **i Note:**
> Due to the above inequalities, the `RCF` check is automatically guaranteed (not considering rounding effects) for these choices of collateral indexes j, whether one is using post maturity mode or not.

After liquidation and repaying a_r we get:

$$D \to D - a_r$$
$$D_m \overset{\approx}{\to} D_m - \gamma_j \cdot a_r$$
$$D_d \overset{\approx}{\to} D_d - a_r$$

It is true that the health ratio can decrease:

$$\frac{D_m}{D} \overset{\approx}{\to} \frac{D_m - \gamma_j \cdot a_r}{D - a_r} < \frac{D_m}{D} < \gamma_j$$

One can even push this value close to 0 by setting $a_r = \frac{c_j \cdot p_j \cdot \beta_j}{\gamma_j}$. And in some special cases where $\frac{c_j \cdot p_j \cdot \beta_j}{\gamma_j} = \frac{D_m}{\gamma_j}$, one can push the health ratio or the max debt to 0. This would just allow consecutive liquidation of the same position since the health factor reduces and thus the position stays unhealthy.

But it is also important to note that the distance between `_position.debt` D and the discounted collateral value (bad-debt gap) of the position D_d stays approximately the same:

$$D - D_d \overset{\approx}{\to} (D - a_r) - (D_d - a_r) = D - D_d$$

This distance is what dictates the value of bad debt since in this context:

$$D^{bad}_{\mathcal{O},u} = [D - D_d]^+$$

Thus, at the liquidation price vector and ignoring rounding, a normal-mode liquidation using `maxLif` preserves the current bad-debt gap $D - D_d$; it does not by itself amplify realized bad debt, even if it worsens the LLTV-based health ratio.

When $\alpha_j = \beta_j = \gamma_j = \gamma_n = 1$ (the special set of market parameters), as soon as the positio becomes unhealthy the above scenario applies.

6.4.2 Footnote

Let

$$D^{max}_{\mathcal{O},u} = \sum_{T \in \mathcal{O}} \left\lfloor \left\lfloor c^T_{\mathcal{O},u} \cdot \frac{p_{\mathcal{O},T}}{10^{18+18}} \right\rfloor \frac{\texttt{lltv}_{\mathcal{O},T}}{10^{18}} \right\rfloor$$

$$D^{dis}_{\mathcal{O},u}(\texttt{lif}) = \sum_{T \in \mathcal{O}} \left\lfloor \left\lceil c^T_{\mathcal{O},u} \cdot \frac{p_{\mathcal{O},T}}{10^{18+18}} \right\rceil \frac{10^{18}}{\texttt{lif}_T} \right\rceil$$

$$D^{bad}_{\mathcal{O},u} = \left[D_{\mathcal{O},u} - D^{dis}_{\mathcal{O},u}(\texttt{lif}^{max}_{\mathcal{O},T}) \right]^+$$

$$D^{capped}_{\mathcal{O},u} = \min\left(D_{\mathcal{O},u}, D^{dis}_{\mathcal{O},u}(\texttt{lif}^{max}_{\mathcal{O},T}) \right)$$

$$a^{max}_{repaid} = \begin{cases} \left\lceil \left(D^{capped}_{\mathcal{O},u} - D^{max}_{\mathcal{O},u} \right) \left\lceil \frac{10^{18}}{10^{18} - \left\lceil \texttt{lif}^{max}_{\mathcal{O},T_j} \cdot \frac{\texttt{lltv}_{\mathcal{O},T_j}}{10^{18}} \right\rceil} \right\rceil \right\rceil & \texttt{lltv}_{\mathcal{O},T_j} < 10^{18} \\ 2^{256} - 1 & \text{(no upper bound)} \end{cases}$$

```
Testing [ LIF x LLTV ] for different LLTV and cursor values:

  LLTV: 0.385      | LIF: 1.181683899556868537      | func: 0.454948301329394387
  LLTV: 0.385      | LIF: 1.444043321299638989      | func: 0.555956678700361011
  LLTV: 0.625      | LIF: 1.103448275862068965      | func: 0.689655172413793104
  LLTV: 0.625      | LIF: 1.230769230769230769      | func: 0.769230769230769231
  LLTV: 0.77      | LIF: 1.061007957559681697      | func: 0.816976127320954907
  LLTV: 0.77      | LIF: 1.129943502824858757      | func: 0.870056497175141243
  LLTV: 0.86      | LIF: 1.036269430051813471      | func: 0.891191709844559586
  LLTV: 0.86      | LIF: 1.075268817204301075      | func: 0.924731182795698925
  LLTV: 0.915      | LIF: 1.021711366538952745      | func: 0.934865900383141762
  LLTV: 0.915      | LIF: 1.044386422976501305      | func: 0.955613577023498695
  LLTV: 0.945      | LIF: 1.01394169835234474      | func: 0.95817490494296578
  LLTV: 0.945      | LIF: 1.028277634961439588      | func: 0.971722365038560411
  LLTV: 0.965      | LIF: 1.008827238335435056      | func: 0.97351828499369483
  LLTV: 0.965      | LIF: 1.017811704834605597      | func: 0.982188295165394402
  LLTV: 0.98      | LIF: 1.005025125628140703      | func: 0.984924623115577889
  LLTV: 0.98      | LIF: 1.010101010101010101      | func: 0.989898989898989899
  LLTV: 1      | LIF: 1      | func: 1
```

```
  LLTV: 1      | LIF: 1      | func: 1


Testing [ 1 / (1 - LIF x LLTV)) ] for different LLTV and cursor values:

  LLTV: 0.385      | LIF: 1.181683899556868537      | func: 1.834688346883468835
  LLTV: 0.385      | LIF: 1.444043321299638989      | func: 2.252032520325203253
  LLTV: 0.625      | LIF: 1.103448275862068965      | func: 3.222222222222222228
  LLTV: 0.625      | LIF: 1.230769230769230769      | func: 4.333333333333333338
  LLTV: 0.77      | LIF: 1.061007957559681697      | func: 5.463768115942028981
  LLTV: 0.77      | LIF: 1.129943502824858757      | func: 7.695652173913043482
  LLTV: 0.86      | LIF: 1.036269430051813471      | func: 9.19047619047619052
  LLTV: 0.86      | LIF: 1.075268817204301075      | func: 13.285714285714285762
  LLTV: 0.915      | LIF: 1.021711366538952745      | func: 15.352941176470588129
  LLTV: 0.915      | LIF: 1.044386422976501305      | func: 22.529411764705882599
  LLTV: 0.945      | LIF: 1.01394169835234474      | func: 23.909090909090909396
  LLTV: 0.945      | LIF: 1.028277634961439588      | func: 35.363636363636363248
  LLTV: 0.965      | LIF: 1.008827238335435056      | func: 37.761904761904762247
  LLTV: 0.965      | LIF: 1.017811704834605597      | func: 56.142857142857142745
  LLTV: 0.98      | LIF: 1.005025125628140703      | func: 66.333333333333331366
  LLTV: 0.98      | LIF: 1.010101010101010101      | func: 99.0000000000000001
  LLTV: 1      | LIF: 1      | func: 0
  LLTV: 1      | LIF: 1      | func: 0


Testing [ LLTV / (1 - LIF x LLTV)) ] for different LLTV and cursor values:

  LLTV: 0.385      | LIF: 1.181683899556868537      | func: 0.706355013550135502
  LLTV: 0.385      | LIF: 1.444043321299638989      | func: 0.867032520325203253
  LLTV: 0.625      | LIF: 1.103448275862068965      | func: 2.013888888888888893
  LLTV: 0.625      | LIF: 1.230769230769230769      | func: 2.708333333333333337
  LLTV: 0.77      | LIF: 1.061007957559681697      | func: 4.207101449275362316
  LLTV: 0.77      | LIF: 1.129943502824858757      | func: 5.925652173913043482
  LLTV: 0.86      | LIF: 1.036269430051813471      | func: 7.903809523809523847
  LLTV: 0.86      | LIF: 1.075268817204301075      | func: 11.425714285714285756
  LLTV: 0.915      | LIF: 1.021711366538952745      | func: 14.047941176470588138
  LLTV: 0.915      | LIF: 1.044386422976501305      | func: 20.614411764705882578
  LLTV: 0.945      | LIF: 1.01394169835234474      | func: 22.594090909090909379
  LLTV: 0.945      | LIF: 1.028277634961439588      | func: 33.418636363636363269
  LLTV: 0.965      | LIF: 1.008827238335435056      | func: 36.440238095238095568
  LLTV: 0.965      | LIF: 1.017811704834605597      | func: 54.177857142857142749
  LLTV: 0.98      | LIF: 1.005025125628140703      | func: 65.006666666666664739
  LLTV: 0.98      | LIF: 1.010101010101010101      | func: 97.020000000000000098
  LLTV: 1      | LIF: 1      | func: 0
  LLTV: 1      | LIF: 1      | func: 0


Testing [ accumulated RCF threshold upperbound ] for different LLTV and cursor values:

  LLTV: 0.385      | LIF: 1.181683899556868537      | func: 2.541043360433604337
  LLTV: 0.385      | LIF: 1.444043321299638989      | func: 3.119065040650406506
  LLTV: 0.625      | LIF: 1.103448275862068965      | func: 5.236111111111111121
  LLTV: 0.625      | LIF: 1.230769230769230769      | func: 7.041666666666666675
  LLTV: 0.77      | LIF: 1.061007957559681697      | func: 9.670869565217391297
  LLTV: 0.77      | LIF: 1.129943502824858757      | func: 13.621304347826086964
  LLTV: 0.86      | LIF: 1.036269430051813471      | func: 17.094285714285714367
  LLTV: 0.86      | LIF: 1.075268817204301075      | func: 24.711428571428571518
  LLTV: 0.915      | LIF: 1.021711366538952745      | func: 29.400882352941176267
  LLTV: 0.915      | LIF: 1.044386422976501305      | func: 43.143823529411765177
  LLTV: 0.945      | LIF: 1.01394169835234474      | func: 46.503181818181818775
  LLTV: 0.945      | LIF: 1.028277634961439588      | func: 68.782272727272726517
  LLTV: 0.965      | LIF: 1.008827238335435056      | func: 74.202142857142857815
```

```
  LLTV: 0.965      | LIF: 1.017811704834605597      | func: 110.320714285714285494
  LLTV: 0.98      | LIF: 1.005025125628140703      | func: 131.339999999999996105
  LLTV: 0.98      | LIF: 1.010101010101010101      | func: 196.020000000000000198
  LLTV: 1      | LIF: 1      | func: 0
  LLTV: 1      | LIF: 1      | func: 0


Testing [ accumulated RCF threshold neg lowerbound ] for different LLTV and cursor values:

  LLTV: 0.385      | LIF: 1.181683899556868537      | func: 3.247398373983739839
  LLTV: 0.385      | LIF: 1.444043321299638989      | func: 3.986097560975609759
  LLTV: 0.625      | LIF: 1.103448275862068965      | func: 7.250000000000000014
  LLTV: 0.625      | LIF: 1.230769230769230769      | func: 9.750000000000000012
  LLTV: 0.77      | LIF: 1.061007957559681697      | func: 13.877971014492753613
  LLTV: 0.77      | LIF: 1.129943502824858757      | func: 19.546956521739130446
  LLTV: 0.86      | LIF: 1.036269430051813471      | func: 24.998095238095238214
  LLTV: 0.86      | LIF: 1.075268817204301075      | func: 36.137142857142857274
  LLTV: 0.915      | LIF: 1.021711366538952745      | func: 43.448823529411764405
  LLTV: 0.915      | LIF: 1.044386422976501305      | func: 63.758235294117647755
  LLTV: 0.945      | LIF: 1.01394169835234474      | func: 69.097272727272728154
  LLTV: 0.945      | LIF: 1.028277634961439588      | func: 102.200909090909089786
  LLTV: 0.965      | LIF: 1.008827238335435056      | func: 110.642380952380953383
  LLTV: 0.965      | LIF: 1.017811704834605597      | func: 164.498571428571428243
  LLTV: 0.98      | LIF: 1.005025125628140703      | func: 196.346666666666660844
  LLTV: 0.98      | LIF: 1.010101010101010101      | func: 293.040000000000000296
  LLTV: 1      | LIF: 1      | func: 0
  LLTV: 1      | LIF: 1      | func: 0
```

6.4.3 Notations

parameter	description
$\mathcal{O}$	a specific obligation/market. Used for indexing. It could be the obligation `id` or the obligation as a whole depending on the context
u	a specific user/position owner
T	a collateral token in the obligation collateral set
T_j	the collateral token seized during liquidation
$c^T_{\mathcal{O},u}$	amount of collateral token T posted by user u in obligation $\mathcal{O}$
$p_{\mathcal{O},T}$	oracle price for collateral token T in obligation $\mathcal{O}$, scaled by `10 ** 36` in the formulas above
$D_{\mathcal{O},u}$	`position[id][u].debt`
$D^{max}_{\mathcal{O},u}$	maximum healthy debt of position $(\mathcal{O}, u)$ under the obligation LLTVs
$D^{dis}_{\mathcal{O},u}$(`lif`)	discounted collateral value of position $(\mathcal{O}, u)$ under liquidation incentive factors `lif`
$D^{bad}_{\mathcal{O},u}$	`badDebt`
$D^{capped}_{\mathcal{O},u}$	debt considered by the recovery close factor calculation, capped at $D^{dis}_{\mathcal{O},u}$(`lif` $^{max}_{\mathcal{O},T}$)
`lltv` $_{\mathcal{O},T}$	loan-to-value threshold for collateral token T in obligation $\mathcal{O}$
`lif` $_{T}$	liquidation incentive factor for collateral token T
`lif` $^{max}_{\mathcal{O},T}$	maximum liquidation incentive factor allowed for collateral token T in obligation $\mathcal{O}$
`lif` $^{max}_{\mathcal{O},-}$	vector of maximum liquidation incentive factors across the collateral tokens in obligation $\mathcal{O}$
a_{repaid}	amount of debt repaid by the liquidator

a_{repaid}^{max}	maximum repayment amount allowed by the recovery close factor formula before threshold slack
a_j	amount of seized collateral token T_j
$\tau_{\mathcal{O}}$	`rcfThreshold` for obligation $\mathcal{O}$
$\tau_{\mathcal{O},T_j}$	token-specific upper-bound slack term for seized token T_j
ϵ	fractional-rounding complement, defined as $1 - \left\{ c_{\mathcal{O},u}^{T_j} \cdot p_{\mathcal{O},T_j} / 10^{36} \right\}$
ϵ_3	rounding-error term used in the less-strict recovery close factor derivation
$F(x, y)$	branch function equal to x when $a_j > 0$ and y when $a_j = 0$
$[x]^+$	positive part of x, equal to $\max(0, x)$