Report

v. 1.0

Customer

Morpho

Smart Contract Audit

pre-liquidation

1st November 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	1.11.24	A. Zveryanskaya	Initial Draft
0.2	1.11.24	A. Zveryanskaya	Minor revision
1.0	1.11.24	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Morpho (formerly known as Morpho Blue) is a trustless and efficient lending primitive with permissionless market creation.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	PreLiquidation.sol	PreLiquidation Factory.sol	
interfaces/			
	IPreLiquidation.sol	IPreLiquidation Callback.sol	IPreLiquidation Factory.sol
libraries/periphery/			
	PreLiquidation AddressLib.sol		
libraries/			
	ErrorsLib.sol	EventsLib.sol	

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We provided Client with a few recommendations.

	Info	Fixed
Moderate	3	0

6 Moderate Issues

CVF-1 INFO

- **Category** Suboptimal
- **Source** PreLiquidationAddressLib.sol

Description This line requires the whole bytecode of the "PreLIquidation" contract to be hashed, which is inefficient.

Recommendation Use precomputed init code hash and pass arguments via callback, as, for example, Uniswap does it: https://github.com/Uniswap/v3-core/blob/main/contracts/UniswapV3Pool.sol#L119

Client Comment *Acknowledged. Implementing this would make the logic much more complex and the improvement is pretty low.*

```
23 keccak256(abi.encodePacked(type(PreLiquidation).creationCode, abi.
     ↪ encode(morpho, id, preLiquidationParams)));
```

CVF-2 INFO

- **Category** Procedural
- **Source** EventsLib.sol

Recommendation The "id" parameter should be indexed.

Client Comment *Acknowledged. It would make event filtering by Id a bit more efficient but it is costly in gas (log2 vs log1) and there should be one pre-liquidation contract per market so not worth implementing.*

```
22 event CreatePreLiquidation(address indexed preLiquidation, Id id,
     ↪ PreLiquidationParams preLiquidationParams);
```

CVF-3 INFO

- **Category** Bad naming
- **Source** IPreLiquidation.sol

Description The semantics of the returned values is unclear.

Recommendation Give descriptive names to the returned values and/or explain in a documentation comment.

Client Comment *We don't like defining values in returns because it leads to weird initialization.*

```
33 returns (uint256, uint256);
```

7 Minor Issues

CVF-4 INFO

- **Category** Procedural
- **Source** PreLiquidationAddressLib.sol

Description We didn't review this file.

```
6 import {Id} from "../../../lib/morpho-blue/src/interfaces/IMorpho.
    ↪ sol";
```

CVF-5 INFO

- **Category** Bad datatype
- **Source** PreLiquidationAddressLib.sol

Recommendation The type for these arguments should be more specific.

Client Comment *Using a generic type is facilitating integrations in other smart contracts.*

```
17 address morpho,
   address factory,
```

CVF-6 INFO

- **Category** Bad datatype
- **Source** PreLiquidationAddressLib.sol

Recommendation The return type should be more specific.

Client Comment *Using a generic type is facilitating integrations in other smart contracts.*

```
21 ) internal pure returns (address) {
```

CVF-7 INFO

- **Category** Procedural
- **Source** EventsLib.sol

Description We didn't review this file.

```
4 import {Id} from "../../lib/morpho-blue/src/interfaces/IMorpho.sol";
```

CVF-8 FIXED

- **Category** Procedural
- **Source** ErrorsLib.sol

Description This version requirement is inconsistent with other files in the same code base.

Recommendation Use consistent version requirements across code base.

Client Comment *Fixed.*

```
2 pragma solidity ^0.8.27;
```

CVF-9 FIXED

- **Category** Procedural
- **Source** ErrorsLib.sol

Recommendation Specify as "^0.8.0" unless there is something special regarding this particular version.

Client Comment *Fixed. (this issue is same as #8)*

```
2 pragma solidity ^0.8.27;
```

CVF-10 INFO

- **Category** Suboptimal
- **Source** ErrorsLib.sol

Recommendation These errors could be made more useful by adding certain parameters to them.

Client Comment *Acknowledged. It's sufficiently clear for us.*

```
11 error PreLltvTooHigh();

13 error PreLCFDecreasing();

15 error PreLCFTooHigh();

17 error PreLIFTooLow();

19 error PreLIFDecreasing();

21 error PreLIFTooHigh();

23 error InconsistentInput();

25 error NotPreLiquidatablePosition();

27 error LiquidatablePosition();

29 error PreLiquidationTooLarge(uint256 repaidShares, uint256
      ↪ repayableShares);

31 error NotMorpho();

33 error NonexistentMarket();
```

CVF-11 INFO

- **Category** Procedural
- **Source** IPreLiquidation.sol

Description Specifying a compiler version range without upper bound is a bad practice, as it is not possible to guarantee compatibility with future major versions.

Recommendation Specify as: ^0.5.0 | ^0.6.0 | ^0.7.0 | ^0.8.0. Also relevant for: IPreLiquidationCallback.sol, IPreLiquidationFactory.sol.

Client Comment *Acknowledged.*

```
2 pragma solidity >= 0.5.0;
```

CVF-12 INFO

- **Category** Procedural
- **Source** IPreLiquidation.sol

Description We didn't review this file.

```
4 import {Id, IMorpho, MarketParams} from "../../lib/morpho-blue/src/
    ↪ interfaces/IMorpho.sol";
```

CVF-13 INFO

- **Category** Suboptimal
- **Source** IPreLiquidation.sol

Description Declaring a top-level structure in a file named after an interface is a bad practice, as it makes it harder navigating through code.

Recommendation Move the structure declaration into the interface or move it into a separate file.

Client Comment *Acknowledged. Implemented like this because it is readable given the size of the code and it is consistent with Morpho Blue.*

```
13 struct PreLiquidationParams {
```

CVF-14 INFO

- **Category** Documentation
- **Source** IPreLiquidation.sol

Description The number format for these fields is unclear.

Recommendation Explain in a documentation comment.

Client Comment *These parameters explained in both the natspec and the readme.*

```
14 uint256 preLltv;
   uint256 preLCF1;
   uint256 preLCF2;
   uint256 preLIF1;
   uint256 preLIF2;
```

CVF-15 INFO

- **Category** Bad datatype
- **Source** IPreLiquidation.sol

Recommendation The type for this field should be more specific.

Client Comment *Using a generic type is facilitating integrations in other smart contracts.*

```
19 address preLiquidationOracle;
```

CVF-16 INFO

- **Category** Procedural
- **Source** IPreLiquidationFactory.sol

Description We didn't review this file.

```
4 import {Id, IMorpho} from "../../lib/morpho-blue/src/interfaces/
    ↪ IMorpho.sol";
```

CVF-17 INFO

- **Category** Documentation
- **Source** IPreLiquidationFactory.sol

Description The semantics of this function is unclear.

Recommendation Add a documentation comment.

Client Comment *Acknowledged. It should be sufficiently clear from the PreLiquidationFactory contract and the isPreLiquidation mapping.*

```
10 function isPreLiquidation(address) external returns (bool);
```

CVF-18 INFO

- **Category** Procedural
- **Source** PreLiquidation.sol

Description This version requirement is inconsistent with other files in the same code base.

Recommendation Use consistent version requirements across code base.

Client Comment *We prefer using an exact version of solidity. Solidity 0.8.27 is needed for custom errors.*

```
2 pragma solidity 0.8.27;
```

CVF-19 INFO

- **Category** Procedural
- **Source** PreLiquidation.sol

Description Specifying a particular compiler version makes it harder migrating to newer versions.

Recommendation Specify as "^0.8.0" or as "^0.8.27" if there is something special regarding this particular version.

Client Comment *We prefer using an exact version of solidity. Solidity 0.8.27 is needed for custom errors.*

```
2 pragma solidity 0.8.27;
```

CVF-20 INFO

- **Category** Procedural
- **Source** PreLiquidation.sol

Description We didn't review these files.

```
4  import {Id, MarketParams, IMorpho, Position, Market} from "../lib/
       ↪ morpho-blue/src/interfaces/IMorpho.sol";
   import {IMorphoRepayCallback} from "../lib/morpho-blue/src/
       ↪ interfaces/IMorphoCallbacks.sol";

8  import {IOracle} from "../lib/morpho-blue/src/interfaces/IOracle.sol
       ↪ ";

10 import {ORACLE_PRICE_SCALE} from "../lib/morpho-blue/src/libraries/
       ↪ ConstantsLib.sol";
   import {SharesMathLib} from "../lib/morpho-blue/src/libraries/
       ↪ SharesMathLib.sol";
   import {SafeTransferLib} from "../lib/solmate/src/utils/
       ↪ SafeTransferLib.sol";
   import {WAD, MathLib} from "../lib/morpho-blue/src/libraries/MathLib
       ↪ .sol";
   import {UtilsLib} from "../lib/morpho-blue/src/libraries/UtilsLib.
       ↪ sol";
   import {ERC20} from "../lib/solmate/src/tokens/ERC20.sol";
```

CVF-21 INFO

- **Category** Bad datatype
- **Source** PreLiquidation.sol

Recommendation The type for these variables should be more specific.

Client Comment *Using a generic type is facilitating integrations in other smart contracts.*

```
36 address internal immutable LOAN_TOKEN;
   address internal immutable COLLATERAL_TOKEN;
   address internal immutable ORACLE;
   address internal immutable IRM;
40 uint256 internal immutable LLTV;

48 address internal immutable PRE_LIQUIDATION_ORACLE;
```

CVF-22 INFO

- **Category** Documentation
- **Source** PreLiquidation.sol

Description The number format for these variables is unclear.

Recommendation Explain in a documentation comment.

Client Comment *These parameters explained in both the natspec and the readme.*

```
43 uint256 internal immutable PRE_LLTV;
   uint256 internal immutable PRE_LCF_1;
   uint256 internal immutable PRE_LCF_2;
   uint256 internal immutable PRE_LIF_1;
   uint256 internal immutable PRE_LIF_2;
```

CVF-23 INFO

- **Category** Bad datatype
- **Source** PreLiquidation.sol

Recommendation The type for the "morpho" argument should be "IMorpho".

Client Comment *Using a generic type is facilitating integrations in other smart contracts.*

```
84 constructor(address morpho, Id id, PreLiquidationParams memory
     ↪ _preLiquidationParams) {
```

CVF-24 INFO

- **Category** Procedural
- **Source** PreLiquidation.sol

Description Names for the returned values are specified in the comment but not in the code.

Recommendation Give names to the returned values in the code.

Client Comment *We don't like defining values in returns because it leads to weird initialization.*

```
122 /// @return seizedAssets The amount of collateral seized.
    /// @return repaidAssets The amount of debt repaid.
```

```
134     returns (uint256, uint256)
```

CVF-25 FIXED

- **Category** Suboptimal
- **Source** PreLiquidation.sol

Description The expression "(ltv - PRE_LLTV).wDivDown(LLTV - PRE_LLTV)" is calculated twice.

Recommendation Calculate once and reuse.

```
153 uint256 preLIF = (ltv - PRE_LLTV).wDivDown(LLTV - PRE_LLTV).wMulDown
    ↪ (PRE_LIF_2 - PRE_LIF_1) + PRE_LIF_1;
```

```
168 uint256 preLCF = (ltv - PRE_LLTV).wDivDown(LLTV - PRE_LLTV).wMulDown
    ↪ (PRE_LCF_2 - PRE_LCF_1) + PRE_LCF_1;
```

CVF-26 INFO

- **Category** Procedural
- **Source** PreLiquidationFactory.sol

Description This version requirement is inconsistent with other files in the same code base.

Recommendation Use consistent version requirements across code base.

Client Comment *Similar to #18*

```
2 pragma solidity 0.8.27;
```

CVF-27 INFO

- **Category** Procedural
- **Source** PreLiquidationFactory.sol

Description Specifying a particular compiler version makes it harder migrating to newer versions.

Recommendation Specify as "^0.8.0" or as "^0.8.27" if there is something special regarding this particular version.

Client Comment *Similar to #18*

```
2 pragma solidity 0.8.27;
```

CVF-28 INFO

- **Category** Procedural
- **Source** PreLiquidationFactory.sol

Description We didn't review these files.

```
6 import {IMorpho, Id} from "../lib/morpho-blue/src/interfaces/IMorpho
    ↪ .sol";
```

CVF-29 INFO

- **Category** Bad datatype
- **Source** PreLiquidationFactory.sol

Recommendation The key type should be "IPreLiquidation".

Client Comment *Using a generic type is facilitating integrations in other smart contracts.*

```
26 mapping(address => bool) public isPreLiquidation;
```

CVF-30 INFO

- **Category** Bad datatype
- **Source** PreLiquidationFactory.sol

Recommendation The argument type should be "IMorpho".

Client Comment *Using a generic type is facilitating integrations in other smart contracts.*

```
31 constructor(address morpho) {
```

CVF-31 INFO

- **Category** Suboptimal
- **Source** PreLiquidationFactory.sol

Description This check is redundant, as it is anyway possible to pass a dead Morpho address.

Recommendation Remove this check.

Client Comment *This is a simple check in case someone called the constructor without any calldata.*

```
32 require(morpho != address(0), ErrorsLib.ZeroAddress());
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting