ABDK CONSULTING

SMART CONTRACT AUDIT

xToken

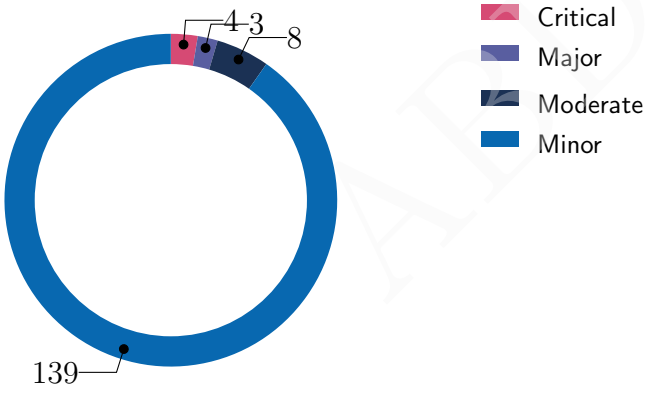
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
25th November 2021

We've been asked to review the 18 files in a Github repo. We found 4 critical, 3 major, and a few less important issues. 3 critical and 3 major issues were fixed.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Opened
CVF-2	Minor	Bad datatype	Opened
CVF-3	Minor	Bad datatype	Opened
CVF-4	Minor	Documentation	Opened
CVF-5	Minor	Documentation	Opened
CVF-6	Minor	Suboptimal	Opened
CVF-7	Major	Overflow/Underflow	Info
CVF-8	Minor	Suboptimal	Opened
CVF-9	Minor	Suboptimal	Opened
CVF-10	Minor	Procedural	Opened
CVF-11	Minor	Bad datatype	Opened
CVF-12	Minor	Bad datatype	Opened
CVF-13	Minor	Suboptimal	Opened
CVF-14	Minor	Bad datatype	Opened
CVF-15	Minor	Bad datatype	Opened
CVF-16	Minor	Bad datatype	Opened
CVF-17	Minor	Bad datatype	Opened
CVF-18	Minor	Procedural	Opened
CVF-19	Minor	Procedural	Opened
CVF-20	Minor	Suboptimal	Opened
CVF-21	Minor	Bad datatype	Opened
CVF-22	Minor	Suboptimal	Opened
CVF-23	Minor	Suboptimal	Opened
CVF-24	Minor	Procedural	Opened
CVF-25	Minor	Suboptimal	Opened
CVF-26	Minor	Suboptimal	Opened
CVF-27	Critical	Flaw	Fixed

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Opened
CVF-29	Minor	Flaw	Opened
CVF-30	Minor	Overflow/Underflow	Opened
CVF-31	Minor	Suboptimal	Opened
CVF-32	Minor	Readability	Opened
CVF-33	Minor	Suboptimal	Opened
CVF-34	Minor	Procedural	Opened
CVF-35	Minor	Bad datatype	Opened
CVF-36	Minor	Procedural	Opened
CVF-37	Minor	Bad datatype	Opened
CVF-38	Minor	Bad naming	Opened
CVF-39	Minor	Procedural	Opened
CVF-40	Minor	Procedural	Opened
CVF-41	Minor	Bad datatype	Opened
CVF-42	Minor	Bad datatype	Opened
CVF-43	Minor	Bad datatype	Opened
CVF-44	Minor	Suboptimal	Opened
CVF-45	Minor	Suboptimal	Opened
CVF-46	Minor	Suboptimal	Opened
CVF-47	Minor	Bad naming	Opened
CVF-48	Minor	Bad datatype	Opened
CVF-49	Minor	Bad datatype	Opened
CVF-50	Minor	Bad datatype	Opened
CVF-51	Minor	Bad datatype	Opened
CVF-52	Minor	Procedural	Opened
CVF-53	Minor	Procedural	Opened
CVF-54	Minor	Unclear behavior	Opened
CVF-55	Minor	Bad datatype	Opened
CVF-56	Minor	Suboptimal	Opened
CVF-57	Minor	Documentation	Opened

ID	Severity	Category	Status
CVF-58	Minor	Documentation	Opened
CVF-59	Minor	Flaw	Opened
CVF-60	Minor	Overflow/Underflow	Opened
CVF-61	Minor	Documentation	Opened
CVF-62	Minor	Suboptimal	Opened
CVF-63	Minor	Suboptimal	Opened
CVF-64	Minor	Suboptimal	Opened
CVF-65	Minor	Suboptimal	Opened
CVF-66	Minor	Suboptimal	Opened
CVF-67	Minor	Suboptimal	Opened
CVF-68	Minor	Suboptimal	Opened
CVF-69	Minor	Suboptimal	Opened
CVF-70	Minor	Suboptimal	Opened
CVF-71	Minor	Suboptimal	Opened
CVF-72	Minor	Suboptimal	Opened
CVF-73	Critical	Flaw	Info
CVF-74	Moderate	Flaw	Opened
CVF-75	Minor	Flaw	Opened
CVF-76	Minor	Procedural	Opened
CVF-77	Moderate	Suboptimal	Opened
CVF-78	Minor	Suboptimal	Opened
CVF-79	Minor	Suboptimal	Opened
CVF-80	Moderate	Unclear behavior	Opened
CVF-81	Minor	Suboptimal	Opened
CVF-82	Minor	Suboptimal	Opened
CVF-83	Minor	Suboptimal	Opened
CVF-84	Minor	Documentation	Opened
CVF-85	Moderate	Suboptimal	Opened
CVF-86	Minor	Documentation	Opened
CVF-87	Minor	Bad datatype	Opened

ID	Severity	Category	Status
CVF-88	Minor	Suboptimal	Opened
CVF-89	Minor	Suboptimal	Opened
CVF-90	Minor	Suboptimal	Opened
CVF-91	Minor	Procedural	Opened
CVF-92	Minor	Procedural	Opened
CVF-93	Minor	Suboptimal	Opened
CVF-94	Minor	Procedural	Opened
CVF-95	Moderate	Suboptimal	Opened
CVF-96	Minor	Procedural	Opened
CVF-97	Moderate	Unclear behavior	Opened
CVF-98	Minor	Flaw	Opened
CVF-99	Minor	Suboptimal	Opened
CVF-100	Minor	Suboptimal	Opened
CVF-101	Minor	Suboptimal	Opened
CVF-102	Moderate	Flaw	Opened
CVF-103	Minor	Suboptimal	Opened
CVF-104	Minor	Suboptimal	Opened
CVF-105	Minor	Suboptimal	Opened
CVF-106	Minor	Bad naming	Opened
CVF-107	Minor	Suboptimal	Opened
CVF-108	Minor	Flaw	Opened
CVF-109	Minor	Bad datatype	Opened
CVF-110	Minor	Bad datatype	Opened
CVF-111	Minor	Procedural	Opened
CVF-112	Minor	Suboptimal	Opened
CVF-113	Major	Suboptimal	Fixed
CVF-114	Minor	Suboptimal	Opened
CVF-115	Minor	Suboptimal	Opened
CVF-116	Minor	Suboptimal	Opened
CVF-117	Minor	Readability	Opened

ID	Severity	Category	Status
CVF-118	Minor	Bad datatype	Opened
CVF-119	Minor	Procedural	Opened
CVF-120	Minor	Overflow/Underflow	Opened
CVF-121	Critical	Flaw	Fixed
CVF-122	Minor	Suboptimal	Opened
CVF-123	Minor	Bad datatype	Opened
CVF-124	Minor	Procedural	Opened
CVF-125	Moderate	Suboptimal	Opened
CVF-126	Minor	Suboptimal	Opened
CVF-127	Minor	Suboptimal	Opened
CVF-128	Minor	Suboptimal	Opened
CVF-129	Minor	Readability	Opened
CVF-130	Minor	Suboptimal	Opened
CVF-131	Critical	Flaw	Fixed
CVF-132	Minor	Procedural	Opened
CVF-133	Minor	Suboptimal	Opened
CVF-134	Minor	Bad datatype	Opened
CVF-135	Minor	Procedural	Opened
CVF-136	Minor	Procedural	Opened
CVF-137	Minor	Bad datatype	Opened
CVF-138	Major	Suboptimal	Fixed
CVF-139	Minor	Procedural	Opened
CVF-140	Minor	Bad datatype	Opened
CVF-141	Minor	Procedural	Opened
CVF-142	Minor	Bad datatype	Opened
CVF-143	Minor	Procedural	Opened
CVF-144	Minor	Bad datatype	Opened
CVF-145	Minor	Procedural	Opened
CVF-146	Minor	Bad naming	Opened
CVF-147	Minor	Bad naming	Opened

ID	Severity	Category	Status
CVF-148	Minor	Documentation	Opened
CVF-149	Minor	Procedural	Opened
CVF-150	Minor	Bad datatype	Opened
CVF-151	Minor	Documentation	Opened
CVF-152	Minor	Suboptimal	Opened
CVF-153	Minor	Documentation	Opened
CVF-154	Minor	Bad naming	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	November 24, 2021	D. Khovratovich	Initial Draft
0.2	November 25, 2021	D. Khovratovich	Minor revision
1.0	November 25, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting 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. We have reviewed the next files:

- interfaces/IComptroller.sol
- interfaces/ILPT.sol
- interfaces/ILiquidityPool.sol
- interfaces/IMarket.sol
- interfaces/IXKNC.sol
- interfaces/IXU3LP.sol
- proxies/ComptrollerProxy.sol
- proxies/LiquidityPoolProxy.sol
- proxies/MarketProxy.sol
- proxies/PriceProxy.sol
- BlockLock.sol
- Comptroller.sol
- LiquidityPool.sol
- LPT.sol
- Market.sol
- Price.sol
- XKNCPrice.sol
- XU3LPPrice.sol

The fixes were provided in the repository.

2.1 About ABDK

ABDK Consulting, 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.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best 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 and that 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 is 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 that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check 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.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Price.sol, XU3LPPrice.sol, XKNCPrice.sol, Market.sol, IComptroller.sol, LPT.sol, LiquidityPool.sol, BlockLock.sol, Comptroller.sol, PriceProxy.sol, MarketProxy.sol, LiquidityPoolProxy.sol, ComptrollerProxy.sol, IXU3LP.sol, IXKNC.sol, IMarket.sol, ILPT.sol, ILiquidityPool.sol.

Recommendation Should be "ĵ0.7.0" according to a common best practice, or "ĵ0.7.3" if there is something special about this particular version.

Listing 1:

```
2 pragma solidity 0.7.3;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Price.sol

Recommendation The type of this variable should be "IERC20".

Listing 2:

```
15 address public underlyingAssetAddress;
```

3.3 CVF-3

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Price.sol

Recommendation The type of these variables should be "AggregatorV3Interface".

Listing 3:

```
16 address public underlyingPriceFeedAddress;
   address public usdcPriceFeedAddress;
```

3.4 CVF-4

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Price.sol

Recommendation The documentation comment for this function should be moved from the implementing contracts to this contract.

Listing 4:

```
22 function getAssetHeld() public view virtual returns (uint256);
```

3.5 CVF-5

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Price.sol

Recommendation The comment is irrelevant and confusing. Consider removing it.

Listing 5:

```
29 uint256 assetHeld = getAssetHeld(); // assetTotalSupply * 1e18
```

3.6 CVF-6

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Price.sol

Recommendation The number of decimals in the underlying and USDC assets could be precomputed. No need to obtain it every time.

Listing 6:

```
32 uint256 assetDecimals = AggregatorV3Interface(
   ↪ underlyingPriceFeedAddress).decimals(); // Depends on the
   ↪ aggregator decimals. Chainlink usually uses 12 decimals

47 uint256 usdcDecimals = AggregatorV3Interface(
   ↪ usdcPriceFeedAddress).decimals();
```

3.7 CVF-7

- **Severity** Major
- **Status** Info
- **Category** Overflow/Underflow
- **Source** Price.sol

Description Overflow is possible here.
Recommendation Consider using safe conversion.
Client Comment The assetHeld has 18 decimals and the priceFeed from chainlink hash 12 decimals. So I don't see the overflow issue here. In case of the overflow happens, it means something went wrong somehow, and will be reverted.

Listing 7:

```
43 .mul(uint256(assetUsdPrice))
```

3.8 CVF-8

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Price.sol

Recommendation The expression "10**(uint256(18).sub(assetDecimals))" could be precomputed. No need to calculate it every time.

Listing 8:

```
45 .mul(10**(uint256(18).sub(assetDecimals)))
```

3.9 CVF-9

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Price.sol

Recommendation The expression "10**(uint256(18).sub(usdcDecimals))" could be precomputed. No need to calculate it every time.

Listing 9:

```
60 .div(10**(uint256(18).sub(usdcDecimals)))
```

3.10 CVF-10

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** XU3LPPrice.sol

Recommendation This variable should be declared as immutable.

Listing 10:

```
12 bool private isToken1PriceFeed;
```

3.11 CVF-11

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** XU3LPPrice.sol

Recommendation The type of this argument should be "IXU3LP".

Listing 11:

```
21 address _underlyingAssetAddress,
```

3.12 CVF-12

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** XU3LPPrice.sol

Recommendation The type of these arguments should be "AggregatorV3Interface".

Listing 12:

```
22 address _underlyingPriceFeedAddress,
   address _usdcPriceFeedAddress,
```

3.13 CVF-13

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** XU3LPPrice.sol

Recommendation These checks are redundant, as it is anyway possible to pass dead addresses as arguments.

Listing 13:

```
26 require(_underlyingAssetAddress != address(0));
   require(_underlyingPriceFeedAddress != address(0));
   require(_usdcPriceFeedAddress != address(0));
```

3.14 CVF-14

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** XKNCPrice.sol

Recommendation The type of this argument should be "IXKNC".

Listing 14:

```
18 address _underlyingAssetAddress,
```

3.15 CVF-15

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** XKNCPrice.sol

Recommendation The type of these arguments should be "AggregatorV3Interface".

Listing 15:

```
19 address _underlyingPriceFeedAddress,
20 address _usdcPriceFeedAddress
```

3.16 CVF-16

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Market.sol

Recommendation The type of this variable should be "Price".

Listing 16:

```
20 address private assetPriceAddress;
```

3.17 CVF-17

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Market.sol

Recommendation The type of this variable should be "IComptroller".

Listing 17:

```
21 address private comptroller;
```

3.18 CVF-18

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Market.sol

Description There is no access level specified for these constants, so internal access will be used by default.
Recommendation Consider explicitly specifying an access level.

Listing 18:

```
27 uint256 constant FACTOR = 1e18;
   uint256 constant PRICE_DECIMALS_CORRECTION = 1e12;
   uint256 constant RATIOS = 1e16;
```

3.19 CVF-19

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Market.sol

Description There is no access level specifying for this mapping, so internal access will be used by default.
Recommendation Consider explicitly specifying access level.

Listing 19:

```
31 mapping(address => uint256) collaterals;
```

3.20 CVF-20

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description Passing the collateral factor as percentage offers quite coarse precision.
Recommendation Consider passing accepting a value with 18 decimals.

Listing 20:

```
42 /// @param _collateralFactor (uint256) collateral factor for
     ↪ this market Ex. 35% should be entered as 35

66 /// @param _collateralFactor (uint256) collateral factor for
     ↪ this market Ex. 35% should be entered as 35
```

3.21 CVF-21

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Market.sol

Recommendation The type of this argument should be "Price".

Listing 21:

```
45 address _assetPriceAddress,
```

3.22 CVF-22

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

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

Listing 22:

```
49 require(_assetPriceAddress != address(0));
```

3.23 CVF-23

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description Returning a value as percentage offers quite coarse precision.
Recommendation Consider returning a raw value with 18 decimals.

Listing 23:

```
60 /// @return (uint256) collateral factor for this market Ex. 35
   ↪ must be understood as 35%
```

3.24 CVF-24

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Market.sol

Description These functions should emit some events.

Listing 24:

```
67 function setCollateralFactor(uint256 _collateralFactor) external
   ↪  override onlyOwner {

79 function setCollateralCap(uint256 _collateralCap) external
   ↪ override onlyOwner {

132 function setComptroller(address _comptroller) external override
   ↪ onlyOwner {

140 function setCollateralizationActive(bool _active) external
   ↪ override onlyOwner {
```

3.25 CVF-25

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description These functions always return true.
Recommendation Consider removing the return values.

Listing 25:

```
84 function pauseContract() external onlyOwner returns (bool) {

90 function unpauseContract() external onlyOwner returns (bool) {
```

3.26 CVF-26

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The expression "IERC20(Price(assetPriceAddress).underlyingAssetAddress())" is calculated twice. Actually, this expression could be precomputed and saved in a storage variable.
Recommendation Consider precomputing this expression or at least calculating it once and reusing.

Listing 26:

```
101     IERC20(Price(assetPriceAddress).underlyingAssetAddress()).
            ↪ balanceOf(address(this)).add(_amount) <=

107 IERC20(Price(assetPriceAddress).underlyingAssetAddress()).
        ↪ transferFrom(msg.sender, address(this), _amount);
```

3.27 CVF-27

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** Market.sol

Description The returned value is ignored.

Listing 27:

```
107 IERC20(Price(assetPriceAddress).underlyingAssetAddress()).
        ↪ transferFrom(msg.sender, address(this), _amount);

157 IERC20(Price(assetPriceAddress).underlyingAssetAddress()).
        ↪ transfer(_liquidator, tokens);

171 IERC20(Price(assetPriceAddress).underlyingAssetAddress()).
        ↪ transfer(msg.sender, _amount);
```

3.28 CVF-28

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Recommendation This function wouldn't be necessary if the "collaterals" mapping would be renaled to "collateral" and made public.

Listing 28:

```
113 function collateral(address _borrower) public view override
    ↪ returns (uint256) {
```

3.29 CVF-29

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Market.sol

Description Multiplication after division is discouraged as it could amplify rounding errors.
Recommendation Consider doing all the multiplications first and then do division once.

Listing 29:

```
124 collaterals[_borrower].mul(assetValueInUSDC).div(
    ↪ PRICE_DECIMALS_CORRECTION).mul(collateralFactor).div(
```

3.30 CVF-30

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** Market.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculations overflow.
Recommendation Consider using the muldiv function described here: https://2π.com/21/muldiv/index.html or some other approach to prevent phantom overflows.

Listing 30:

```
124     collaterals[_borrower].mul(assetValueInUSDC).div(
        ↪ PRICE_DECIMALS_CORRECTION).mul(collateralFactor).div(

154 uint256 tokens = _amount.mul(PRICE_DECIMALS_CORRECTION).div(
    ↪ Price(assetPriceAddress).getPrice());
```

3.31 CVF-31

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Recommendation This check is redundant as it is anyway possible to specify a dead comptroller address.

Listing 31:

```
133 require(_comptroller != address(0));
```

3.32 CVF-32

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Market.sol

Description Here the "SafeMath.sub" function is used to enforce a business-level constraint. This makes code harder to read and less reliable.
Recommendation Consider explicitly checking business level constraints.

Listing 32:

```
156 collaterals[_borrower] = collaterals[_borrower].sub(tokens);
```

3.33 CVF-33

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Market.sol

Description The expression "collaterals[msg.sender]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 33:

```
164 require(collaterals[msg.sender] >= _amount, "You have not
    ↪ collateralized that much");

170 collaterals[msg.sender] = collaterals[msg.sender].sub(_amount);
```

3.34 CVF-34

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IComptroller.sol

Recommendation Documentation comments for the functions defined in this interface should be moved from the implementing smart contract to the interface itself.

Listing 34:

```
4 interface IComptroller {
```

3.35 CVF-35

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** IComptroller.sol

Recommendation The argument type should be "IMarket".

Listing 35:

```
5 function addMarket(address _market) external;
```

3.36 CVF-36

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IComptroller.sol

Recommendation This function should emit some event and this event should be defined in this interface.

Listing 36:

```
5 function addMarket(address _market) external;
```

3.37 CVF-37

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** IComptroller.sol

Recommendation The argument type should be "ILiquidityPool".

Listing 37:

```
7 function setLiquidityPool(address _liquidityPool) external;
```

3.38 CVF-38

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** IComptroller.sol

Recommendation The word "comptroller" is an acronym for "compound controller". As this protocol is not named "compound" consider choosing a different name.

Listing 38:

```
4 interface IComptroller {
```

3.39 CVF-39

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IComptroller.sol

Recommendation This function should emit some event and this event should be defined in this interface.

Listing 39:

```
7 function setLiquidityPool(address _liquidityPool) external;
```

3.40 CVF-40

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IComptroller.sol

Description Indicating errors by return value is discouraged.
Recommendation Consider reverting in case of error. Those, who need to handle errors, could always use a try/catch block.

Listing 40:

```
17 ) external returns (bool);

24 ) external returns (bool);
```

3.41 CVF-41

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** IComptroller.sol

Recommendation The type of this argument should be "IMarket[] memory _markets" or even "IMarket[] calldata _markets".

Listing 41:

```
23 address[] memory _markets
```

3.42 CVF-42

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LPT.sol

Recommendation The type of this variable should be "ILiquidityPool".

Listing 42:

```
11 address public liquidityPool;
```

3.43 CVF-43

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LPT.sol

Recommendation The type of this argument should be "ILiquidityPool".

Listing 43:

```
21 address _liquidityPool
```

3.44 CVF-44

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LPT.sol

Description This check is redundant, it is anyway possible to specify a dead liquidity pool address.

Listing 44:

```
23 require(_liquidityPool != address(0));
```

3.45 CVF-45

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LPT.sol

Recommendation It would be more efficient to override the "decimals" function and return a constant value from it.

Listing 45:

```
24 _setupDecimals(6);
```

3.46 CVF-46

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LPT.sol

Description These functions always return true.
Recommendation Consider returning nothing.

Listing 46:

```
38 function mint(address _recipient, uint256 _amount) external
   ↪ override onlyLiquidityPool returns (bool) {

47 function burnFrom(address _sender, uint256 _amount) external
   ↪ override onlyLiquidityPool returns (bool) {
```

3.47 CVF-47

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** LiquidityPool.sol

Description It is unclear from the field name that this value is actually a block number.
Recommendation Consider renaming to "borrowedAtBlock".

Listing 47:

```
23 uint256 borrowedAt;
```

3.48 CVF-48

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LiquidityPool.sol

Recommendation The type of this variable should be "IERC20".

Listing 48:

```
26 address private stableCoin;
```

3.49 CVF-49

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LiquidityPool.sol

Recommendation The type of this variable should be "ILPT".

Listing 49:

```
27 address private liquidityPoolToken;
```

3.50 CVF-50

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LiquidityPool.sol

Recommendation This variable should be declared as immutable.

Listing 50:

```
26 address private stableCoin;
```

3.51 CVF-51

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LiquidityPool.sol

Recommendation The type of this variable should be "IComptroller".

Listing 51:

```
41 address public comptroller;
```

3.52 CVF-52

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidityPool.sol

Description There is no access level specified for this mapping, so internal access will be used by default.
Recommendation Consider explicitly specifying an access level.

Listing 52:

```
49 mapping(address => Borrow) borrows;
```

3.53 CVF-53

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidityPool.sol

Description There are no access level specified for these constants, so internal access will be used by default.
Recommendation Consider explicitly specifying an access level.

Listing 53:

```
51 uint256 constant RATIOS = 1e16;
uint256 constant FACTOR = 1e18;
uint256 constant BLOCKS_PER_YEAR = 2628000;
```

3.54 CVF-54

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** LiquidityPool.sol

Description Relying on a particular block rate is a bad practice, as the rate could change over time.
Recommendation Consider measuring time intervals in seconds and relying on block timestamps.

Listing 54:

```
53 uint256 constant BLOCKS_PER_YEAR = 2628000;
```

3.55 CVF-55

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LiquidityPool.sol

Recommendation The argument type should be "IERC20".

Listing 55:

```
57 function initialize(address _stableCoin) external initializer {
```

3.56 CVF-56

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation This check is redundant. It is anyway possible to specify a dead stable coin address.

Listing 56:

```
58 require(_stableCoin != address(0));
```

3.57 CVF-57

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidityPool.sol

Recommendation This comment is confusing as the stable coin is not necessary USDC.

Listing 57:

```
66 /// @notice USDC owned by this Liquidity Pool
   /// @return (uint256) How much USDC the Liquidity Pool owns

102 /// @return (uint256) How much a Borrower owes to the Liquidity
      ↪ Pool in USDC terms

169 /// @notice Lenders can supply as much USDC as they want into
      ↪ the Liquidity Pool

171 /// @param _amount (uint256) Amount of USDC to be supplied into
      ↪ the Liquidity Pool

180 /// @notice Lenders can exchange their LPT for USDC upon interes
      ↪  earned by the protocol
   /// @dev This will burn LPT in exchange for USDC

199 /// @notice Borrowers can borrow USDC having their collaterals
      ↪ as guarantee
```

3.58 CVF-58

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidityPool.sol

Description The actual return value is the utilization rate multiplied by FACTOR.
Recommendation Consider reflecting this fact in the documentation comment.

Listing 58:

```
73 /// @return (uint256) Utilization Rate value
```

3.59 CVF-59

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** LiquidityPool.sol

Description This reverts in case the denominator is not positive.
Recommendation Consider returning some special value in this case instead of reverting. Otherwise, inability to calcualte the utilization rate could make many protocol use cases completely non-operational.

Listing 59:

```
76 return totalBorrows.mul(FACTOR).div(totalBorrows.add(
   ↪ currentLiquidity()).sub(reserves).sub(xtkEarns));
```

3.60 CVF-60

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** LiquidityPool.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculations overflow.
Recommendation Consider using the muldiv function described here: https://2π.com/21/muldiv/index.html or some other approach to prevent phantom overflows.

Listing 60:

```
76  return totalBorrows.mul(FACTOR).div(totalBorrows.add(
      ↪ currentLiquidity()).sub(reserves).sub(xtkEarns));

85      return slope1.mul(utilizationRate()).div(FACTOR).add(
          ↪ baseBorrowRate);

87      baseBorrowRate.add(slope1.mul(optimalUtilizationRate).div(
          ↪ FACTOR)).add(
            slope2.mul(utilizationRate().sub(optimalUtilizationRate)
              ↪ ).div(FACTOR)

109 return borrowerBorrow.amount.mul(newBorrowIndex).div(
      ↪ borrowerBorrow.interestIndex);

138 newBorrowIndex = borrowIndex.mul(interestFactor).div(FACTOR).add
      ↪ (borrowIndex);
    newTotalBorrow = totalBorrows.mul(interestFactor).div(FACTOR).
      ↪ add(totalBorrows);

150     .mul(reserveFactor)
        .div(FACTOR)
        .mul(totalBorrows)
        .div(FACTOR);

163 uint256 xtkInterest = borrowRatePerBlock().mul(deltaBlock).mul(
      ↪ xtkFeeFactor).div(FACTOR).mul(totalBorrows).div(

177 ILPT(liquidityPoolToken).mint(msg.sender, _amount.mul(
      ↪ currentLptPrice).div(FACTOR));

188 uint256 usdcAmount = _lptAmount.mul(FACTOR).div(currentLptPrice)
      ↪ ;

193     finalLPTAmount = finalAmount.mul(currentLptPrice).div(FACTOR
          ↪ );
```

(... 310, 455)

3.61 CVF-61

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidityPool.sol

Description The actual return values are multiplied by FACTOR.
Recommendation Consider reflecting this fact in the documentation comments.

Listing 61:

```
82 /// @return (uint256) Borrow rate value

94 /// @return (uint256) Borrow rate per block value
```

3.62 CVF-62

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The expression "utilizationRate()" is calculated twice here.
Recommendation Consider calculating one and reusing.

Listing 62:

```
84 if (utilizationRate() <= optimalUtilizationRate)
       return slope1.mul(utilizationRate()).div(FACTOR).add(
          ↪ baseBorrowRate);

88         slope2.mul(utilizationRate().sub(optimalUtilizationRate)
              ↪ ).div(FACTOR)
```

3.63 CVF-63

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation The expression "baseBorrowRate.add(slope1.mul(optimalUtilizationRate).div(FACTOR))" could be precalculated and saved in the storage, as this expression doesn't depend on the protocol state.

Listing 63:

```
87 baseBorrowRate.add(slope1.mul(optimalUtilizationRate).div(FACTOR
       ↪ )).add(
```

3.64 CVF-64

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation This expression could be refactored to perform division by FACTOR only once: baseBorrowRate.add(slope1.mul(optimalUtilizationRate).add(slope2.mul(utilizationRate().sub(optimalUtilizationRate))).div(FACTOR))

Listing 64:

```
87 baseBorrowRate.add(slope1.mul(optimalUtilizationRate).div(FACTOR
    ↪ )).add(
     slope2.mul(utilizationRate().sub(optimalUtilizationRate)).
        ↪ div(FACTOR)
);
```

3.65 CVF-65

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation It would be more logical to calculate a per-block or even a per-second rate first, using the base rate, the optimal rate and the slopes, and then derive the annual rate by raising the per-block or per-second rate to the power of the number of blocks or seconds per year.

Listing 65:

```
96 return borrowRate().div(BLOCKS_PER_YEAR);
```

3.66 CVF-66

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The "borrowerBorrow.amount" expression is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 66:

```
106 if (borrowerBorrow.amount == 0) return 0;

109 return borrowerBorrow.amount.mul(newBorrowIndex).div(
    ↪ borrowerBorrow.interestIndex);
```

3.67 CVF-67

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation This variable is redundant, as "block.number" is cheaper to access than a local variable.

Listing 67:

```
115 uint256 currentBlock = block.number;
```

3.68 CVF-68

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The following value is calculated inside all three calls here: borrowRatePerBlock * (currentBlock - accrualBlock) * totalBorrows This is suboptimal.
Recommendation Consider refactoring the code to calculate this value only once.

Listing 68:

```
116 reserves = calculateReservesInformation(currentBlock);
    xtkEarns = calculateXtkEarnings(currentBlock);
    (borrowIndex, totalBorrows) = calculateBorrowInformationAtBlock(
       ↪ currentBlock);
```

3.69 CVF-69

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation It should be 'porrowRatePerBlock+1' raised to the power of deltaBlock and then reduced by one. Raising to an integer power is quite cheap.

Listing 69:

```
136 uint256 interestFactor = borrowRatePerBlock().mul(deltaBlock);

148 uint256 reservesInterest = borrowRatePerBlock()
        .mul(deltaBlock)

163 uint256 xtkInterest = borrowRatePerBlock().mul(deltaBlock).mul(
       ↪ xtkFeeFactor).div(FACTOR).mul(totalBorrows).div(
```

3.70 CVF-70

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation The final additions would be not necessary if FACTOR would be added to the "interestFactor" before the multiplication.

Listing 70:

```
138 newBorrowIndex = borrowIndex.mul(interestFactor).div(FACTOR).add
    ↪ (borrowIndex);
newTotalBorrow = totalBorrows.mul(interestFactor).div(FACTOR).
    ↪ add(totalBorrows);
```

3.71 CVF-71

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description Unlike in the "calculateBorrowInformationAtBlock" function, these functions don't handle situation when the "_block" value is less than the accrual block.
Recommendation Consider handling this situation for consistency.

Listing 71:

```
147 uint256 deltaBlock = _block.sub(accrualBlock);

162 uint256 deltaBlock = _block.sub(accrualBlock);
```

3.72 CVF-72

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description Multiplication after division is discouraged, as it could amplify rounding errors.
Recommendation Consider doing division after all the multiplications.

Listing 72:

```
151       .div(FACTOR)
          .mul(totalBorrows)

163 uint256 xtkInterest = borrowRatePerBlock().mul(deltaBlock).mul(
    ↪ xtkFeeFactor).div(FACTOR).mul(totalBorrows).div(
      FACTOR
```

3.73 CVF-73

- **Severity** Critical
- **Status** Info
- **Category** Flaw
- **Source** LiquidityPool.sol

Description The returned value is ignored here.

Listing 73:

```
176 IERC20(stableCoin).transferFrom(msg.sender, address(this),
    ↪ _amount);
ILPT(liquidityPoolToken).mint(msg.sender, _amount.mul(
    ↪ currentLptPrice).div(FACTOR));

195 ILPT(liquidityPoolToken).burnFrom(msg.sender, finalLPTAmount);
IERC20(stableCoin).transfer(msg.sender, finalAmount);

223 IERC20(stableCoin).transfer(msg.sender, _amount);

243 IERC20(stableCoin).transferFrom(msg.sender, address(this),
    ↪ _amount);

306 IERC20(stableCoin).transferFrom(msg.sender, address(this),
    ↪ _amount);

321 IERC20(stableCoin).transfer(_recipient, xtkEarns);
```

3.74 CVF-74

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** LiquidityPool.sol

Description In case the liquidity pool token is not set, this call will effectively does nothing, however stable coins will still be taken from the caller.
Recommendation Consider explicitly requiring the liquidity pool token to be set here.

Listing 74:

```
177 ILPT(liquidityPoolToken).mint(msg.sender, _amount.mul(
    ↪ currentLptPrice).div(FACTOR));
```

3.75 CVF-75

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** LiquidityPool.sol

Description This rounds the LPT amount down, i.e. towards the user.
Recommendation Consider rounding towards the protocol to prevent possible abuse.

Listing 75:

```
193 finalLPTAmount = finalAmount.mul(currentLptPrice).div(FACTOR);
```

3.76 CVF-76

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidityPool.sol

Recommendation These functions should emit some events to make it easier for liquidators to track debts.

Listing 76:

```
204 function borrow(uint256 _amount) external notLocked(msg.sender)
      ↪ whenNotPaused {

230 function repay(uint256 _amount) public notLocked(msg.sender)
      ↪ whenNotPaused {

260 function liquidate(address _borrower, uint256 _amount) external
      ↪ notLocked(msg.sender) whenNotPaused {

271 function liquidateWithPreference(
```

3.77 CVF-77

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description This check is redundant in case the user already owes more than the minimumLoanValue and just adds more debt on top of it. This check is also insufficient, as the user may borrow more that the minimumLoanValue and then immediately repay the debt partially, so the remaining debt will be below the minimum value.
Recommendation Consider removing this check and putting another check at the very end of the function to ensure that the debt after executing this function is above the minimum value.

Listing 77:

```
205 require(_amount >= minimumLoanValue, "You must borrow the
    ↪ minimum loan value or more");
```

3.78 CVF-78

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The expression "borrowerBorrow.amount" is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 78:

```
211 borrowerBorrow.amount = updatedBorrowBy(msg.sender);

214     IComptroller(comptroller).borrowingCapacity(msg.sender).sub(
        ↪ borrowerBorrow.amount) >= _amount,

218 borrowerBorrow.amount = borrowerBorrow.amount.add(_amount);
```

3.79 CVF-79

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The "borrowerBorrow.amount" field is assigned twice.
Recommendation Consider calculating the new value in a local variable and then assigning once.

Listing 79:

```
211 borrowerBorrow.amount = updatedBorrowBy(msg.sender);

218 borrowerBorrow.amount = borrowerBorrow.amount.add(_amount);
```

3.80 CVF-80

- **Severity** Moderate
- **Status** Opened
- **Category** Unclear behavior
- **Source** LiquidityPool.sol

Recommendation This function should ensure that the debt after the function execution is either zero or not below the "minimumLoanValue" threshold. Otherwise it would be possible to borrow a large debt and then immediately repay a part of it, so the remaining debt will be below the minimum value.

Listing 80:

```
230 function repay(uint256 _amount) public notLocked(msg.sender)
    ↪ whenNotPaused {
```

3.81 CVF-81

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The "borrowerBorrow.amount" field is assigned twice.
Recommendation Consider calculating the new value in a local variable and then assigning once.

Listing 81:

```
236 borrowerBorrow.amount = updatedBorrowBy(msg.sender);

245 borrowerBorrow.amount = borrowerBorrow.amount.sub(_amount);
```

3.82 CVF-82

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The "borrowerBorrow.amount" expression is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 82:

```
239 require(borrowerBorrow.amount > 0, "You have no borrows to be
    ↪ repaid");

241 if (_amount > borrowerBorrow.amount) _amount = borrowerBorrow.
    ↪ amount;

245 borrowerBorrow.amount = borrowerBorrow.amount.sub(_amount);
```

3.83 CVF-83

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description Calling external contracts before updating state is discouraged as it potentially could be used for reentrancy attacks.
Recommendation Consider moving the "transferFrom" call to teh very end of the function.

Listing 83:

```
243 IERC20(stableCoin).transferFrom(msg.sender, address(this),
    ↪ _amount);

245 borrowerBorrow.amount = borrowerBorrow.amount.sub(_amount);
    totalBorrows = totalBorrows.sub(_amount);

306 IERC20(stableCoin).transferFrom(msg.sender, address(this),
    ↪ _amount);
```

3.84 CVF-84

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidityPool.sol

Recommendation This should be ‘Borrow amount‘.

Listing 84:

```
259 /// @param _amount (address) Borrower's address

269 /// @param _amount (address) Borrower's address
```

3.85 CVF-85

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description Locking the liquidator account could make liquidate less efficient. It is normal to run a robot that will perform liquidations automatically, and in case of sharp market moves, many accounts could become eligible for liquidation at once, so the robot will either have to wait between liquidations, which would increase the probability that some account will not be liquidated fast enough and will go underwater; or the robot will have to use several accounts and continuously move tokens between then, which will make the liquidation costs higher.
Recommendation Consider removing these locks.

Listing 85:

```
260 function liquidate(address _borrower, uint256 _amount) external
      ↪ notLocked(msg.sender) whenNotPaused {
        lock(msg.sender);

275 ) external notLocked(msg.sender) whenNotPaused {
        lock(msg.sender);
```

3.86 CVF-86

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** LiquidityPool.sol

Recommendation This comment is incorrect.

Listing 86:

```
270 /// @param _markets (address) Borrower's address
```

3.87 CVF-87

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LiquidityPool.sol

Recommendation The type of this argument should be "IMarket[] memory".

Listing 87:

```
274 address[] memory _markets

288 address[] memory _markets
```

3.88 CVF-88

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The "borrowerBorrow.amount" value is read from the storage twice just after being written there.
Recommendation Consider cashing the just written value in a local variable and reusing.

Listing 88:

```
296 borrowerBorrow.amount = updatedBorrowBy(_borrower);

304 if (_amount > borrowerBorrow.amount) _amount = borrowerBorrow.
    ↪ amount;

307 borrowerBorrow.amount = borrowerBorrow.amount.sub(_amount);
```

3.89 CVF-89

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description The "borrowerBorrow.amount" field is assigned twice.
Recommendation Consider calculating the new value in a local variable and then assigning once.

Listing 89:

```
296 borrowerBorrow.amount = updatedBorrowBy(_borrower);

307 borrowerBorrow.amount = borrowerBorrow.amount.sub(_amount);
```

3.90 CVF-90

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation The value "1e18.sub(liquidityPenaltyFactor" could be precomputed and stored in a storage variable, as this value doesn't depend of the protocol state.

Listing 90:

```
310 uint256 amount = _amount.mul(FACTOR).div(uint256(1e18).sub(
    ↪ liquidityPenaltyFactor));
```

3.91 CVF-91

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidityPool.sol

Recommendation This function should log some event in the case xtkEarns is nonzero.

Listing 91:

```
319 function withdrawFees(address _recipient) external onlyOwner {
```

3.92 CVF-92

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidityPool.sol

Recommendation This function should emit some event. It is also unclear whether it is fine to call this function several times.

Listing 92:

```
327 function setLiquidityPoolToken(address _liquidityPoolToken)
    ↪ external onlyOwner {
```

3.93 CVF-93

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation This check is redundant, as it is anyway possible to pass a dead liquidity pool token address.

Listing 93:

```
328 require(_liquidityPoolToken != address(0));
```

3.94 CVF-94

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidityPool.sol

Recommendation This function should emit some event. It is also unclear whether it is fine to call this function several times.

Listing 94:

```
334 function setComptroller(address _comptroller) external onlyOwner
    ↪ {
```

3.95 CVF-95

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description Entering the parameters as percentages offers very coarse precision.
Recommendation Consider accepting the parameters as fixed point number with 18 decimals.

Listing 95:

```
339 /// @dev This parameters must be entered as percentages. Ex 35
    ↪ is meant to be understood as 35%

357 /// @dev This parameter must be entered as percentage. Ex 35 is
    ↪ meant to be understood as 35%

364 /// @dev This parameter must be entered as percentage. Ex 35 is
    ↪ meant to be understood as 35%

383 /// @dev This parameter must be entered as percentage. Ex 35 is
    ↪ meant to be understood as 35%
```

3.96 CVF-96

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidityPool.sol

Recommendation This function should emit some event.

Listing 96:

```
344 function setInterestModelParameters(

359 function setReserveFactor(uint256 _reserveFactor) external
    ↪ onlyOwner {

366 function setXtkFeeFactor(uint256 _xtkFeeFactor) external
    ↪ onlyOwner {

372 function setLPTBaseValue(uint256 _lptBaseValue) external
    ↪ onlyOwner {

378 function setMinimumLoanValue(uint256 _minimumLoanValue) external
    ↪  onlyOwner {

385 function setLiquidationPenaltyFactor(uint256
    ↪ _liquidityPenaltyFactor) external onlyOwner {
```

3.97 CVF-97

- **Severity** Moderate
- **Status** Opened
- **Category** Unclear behavior
- **Source** LiquidityPool.sol

Recommendation This function should call "accrueInterest" before applying the changes. Otherwise, the changes would have retrospective effect.

Listing 97:

```
344 function setInterestModelParameters(

359 function setReserveFactor(uint256 _reserveFactor) external
    ↪ onlyOwner {

366 function setXtkFeeFactor(uint256 _xtkFeeFactor) external
    ↪ onlyOwner {
```

3.98 CVF-98

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** LiquidityPool.sol

Description There are no range checks for these argument.
Recommendation Consider adding relevant checks.

Listing 98:

```
345     uint256 _optimalUtilizationRate,
        uint256 _baseBorrowRate,
        uint256 _slope1,
        uint256 _slope2

359 function setReserveFactor(uint256 _reserveFactor) external
      ↪ onlyOwner {

366 function setXtkFeeFactor(uint256 _xtkFeeFactor) external
      ↪ onlyOwner {

372 function setLPTBaseValue(uint256 _lptBaseValue) external
      ↪ onlyOwner {

378 function setMinimumLoanValue(uint256 _minimumLoanValue) external
      ↪  onlyOwner {

385 function setLiquidationPenaltyFactor(uint256
      ↪ _liquidityPenaltyFactor) external onlyOwner {
```

3.99 CVF-99

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description These functions always return true.
Recommendation Consider removing the return value.

Listing 99:

```
390 function pauseContract() external onlyOwner returns (bool) {

392     return true;

396 function unpauseContract() external onlyOwner returns (bool) {

398     return true;
```

3.100 CVF-100

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Description Returning values as percentages offers very coarse precision.
Recommendation Consider returning raw fixed-point values with 18 decimals.

Listing 100:

```
402 /// @dev This parameter must be understood as a percentage. Ex
      ↪ 35 is meant to be understood as 35%

409 /// @dev This parameter must be understood as a percentage. Ex
      ↪ 35 is meant to be understood as 35%

416 /// @dev This parameter must be understood as a percentage. Ex
      ↪ 35 is meant to be understood as 35%

423 /// @dev This parameter must be understood as a percentage. Ex
      ↪ 35 is meant to be understood as 35%

430 /// @dev This parameter must be understood as a percentage. Ex
      ↪ 35 is meant to be understood as 35%

437 /// @dev This parameter must be understood as a percentage. Ex
      ↪ 35 is meant to be understood as 35%

465 /// @dev This parameter must be understood as a percentage. Ex
      ↪ 35 is meant to be understood as 35%
```

3.101 CVF-101

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** LiquidityPool.sol

Recommendation These functions wouldn't be necessary if the corresponding variable would be declared as public.

Listing 101:

```
404 function getOptimalUtilizationRate() external view returns (
    ↪ uint256) {

411 function getBaseBorrowRate() external view returns (uint256) {

418 function getSlope1() external view returns (uint256) {

425 function getSlope2() external view returns (uint256) {

439 function getXtkFeeFactor() external view returns (uint256) {

445 function getLPTBaseValue() external view returns (uint256) {

460 function getMinimumLoanValue() external view returns (uint256) {

467 function getLiquidationPenaltyFactor() external view returns (
    ↪ uint256) {
```

3.102 CVF-102

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** LiquidityPool.sol

Description Here potentially outdated values of "totalBorrows", "reserves", and "xtkEarns" are used.
Recommendation Consider using up to date values.

Listing 102:

```
455 totalSupplyLiquidityPool.mul(FACTOR).div(currentLiquidity().add(
    ↪ totalBorrows).sub(reserves).sub(xtkEarns));
```

3.103 CVF-103

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BlockLock.sol

Description Measuring time in blocks is discouraged, as block rate could change over time.
Recommendation Consider measuring time in seconds.

Listing 103:

```
9  // how many blocks are the functions locked for
10 uint256 private constant BLOCK_LOCK_COUNT = 16;
```

3.104 CVF-104

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BlockLock.sol

Description The "_address" argument equals to "msg.sender" in all existing usages.
Recommendation Consider just using "msg.sender" and remove the argument.

Listing 104:

```
14 function lock(address _address) internal {
```

3.105 CVF-105

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BlockLock.sol

Description This function and modifier are always used together: a function market with the modifier always performs lock.
Recommendation Consider merging the function and the modifier together into a single modifier that first ensures that no lock is set for "msg.sender" and then sets the lock.

Listing 105:

```
14 function lock(address _address) internal {

18 modifier notLocked(address lockedAddress) {
```

3.106 CVF-106

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** BlockLock.sol

Description The modifier name is confusing. In a function signature it will look like: function foo () notLocked {} so one could thing that this function is not locked, while it is actually locked (at least potentially).
Recommendation Consider renaming to "lockable" or "onlyWhenNotLocked" or something like this.

Listing 106:

```
18 modifier notLocked(address lockedAddress) {
```

3.107 CVF-107

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** BlockLock.sol

Description The "lockedAddress" argument equals to "msg.sender" in all existing usages.
Recommendation Consider just using "msg.sender" in the modifier body and remove the argument.

Listing 107:

```
18 modifier notLocked(address lockedAddress) {
```

3.108 CVF-108

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** BlockLock.sol

Recommendation Should be ‘<‘ if ‘lastLockedBlock‘ is locked.

Listing 108:

```
19 require(lastLockedBlock[lockedAddress] <= block.number);
```

3.109 CVF-109

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Comptroller.sol

Recommendation The type for this variable should be "ILiquidityPool".

Listing 109:

```
17 address public liquidityPool;
```

3.110 CVF-110

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Comptroller.sol

Recommendation The type for this variable should be "IMarket[]".

Listing 110:

```
19 address[] public markets;
```

3.111 CVF-111

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Comptroller.sol

Description There is no access level specified for these constants, so internal access will be used by default.
Recommendation Consider explicitly specifying access level.

Listing 111:

```
21 uint256 constant RATIOS = 1e16;
   uint256 constant FACTOR = 1e18;
```

3.112 CVF-112

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description The error message is too generic and too long at the same time.
Recommendation Consider making it shorter and more specific.

Listing 112:

```
26 require(msg.sender == liquidityPool, "You are not allowed to
   ↪ perform this action");
```

3.113 CVF-113

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** Comptroller.sol

Description This allows adding the same market several times, thus the collateral at this market will be counted several times too.
Recommendation Consider forbidding adding a market that is already added.

Listing 113:

```
38 function addMarket(address _market) external override onlyOwner
   ↪ {
```

3.114 CVF-114

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description This check is redundant. Why not to know the borrowing capacity of the zero address?

Listing 114:

```
53 require(_borrower != address(0));
```

3.115 CVF-115

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description The "capacity" variable is used before initialization.
Recommendation Consider explicitly initializing to zero for readability.

Listing 115:

```
55 capacity = capacity.add(IMarket(markets[i]).borrowingLimit(
   ↪ _borrower));
```

3.116 CVF-116

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description The "FACTOR" constant is confusing here. It is supposed to be "one" value with 18 decimals, but here it is used as an "infinity" value with two decimals.
Recommendation Consider defining separate constant for indefinitely large health ratio.

Listing 116:

```
65 if (currentBorrow == 0) return FACTOR;
```

3.117 CVF-117

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Comptroller.sol

Recommendation Should be "else return" for readability.

Listing 117:

```
66 return borrowingCapacity(_borrower).mul(1e2).div(currentBorrow);
```

3.118 CVF-118

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Comptroller.sol

Recommendation The value "1e2" should be turned into a named constant.

Listing 118:

```
66 return borrowingCapacity(_borrower).mul(1e2).div(currentBorrow);
```

3.119 CVF-119

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Comptroller.sol

Description Using only two decimals for health ratio is quite coarse.
Recommendation Consider using more decimals. 18 decimals is de-facto standard.

Listing 119:

```
66 return borrowingCapacity(_borrower).mul(1e2).div(currentBorrow);
```

3.120 CVF-120

- **Severity** Minor
- **Status** Opened
- **Category** Overflow/Underflow
- **Source** Comptroller.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculations overflow.
Recommendation Consider using the muldiv function described here: https://2π.com/21/muldiv/index.html or some other approach to prevent phantom overflows.

Listing 120:

```
66 return borrowingCapacity(_borrower).mul(1e2).div(currentBorrow);

103     uint256 collateral = borrowingLimit.mul(FACTOR).div(
          ↪ collateralFactor);

129     uint256 collateral = borrowingLimit.mul(FACTOR).div(
          ↪ collateralFactor);
```

3.121 CVF-121

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** Comptroller.sol

Description This function doesn't have any return statements, so it always returns false, even on successful executions.

Listing 121:

```
77 /// @return (bool) Indicates a successful operation
```

3.122 CVF-122

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Recommendation These checks are redundant. It is anyway possible to pass invalid addresses.

Listing 122:

```
83 require(_liquidator != address(0));
   require(_borrower != address(0));
```

3.123 CVF-123

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Comptroller.sol

Recommendation The type of this variable should be "IMarket[] memory".

Listing 123:

```
85 address[] memory localMarkets = markets;
```

3.124 CVF-124

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Comptroller.sol

Description These variables are used before initialization.
Recommendation Consider explicitly initializing with zero values.

Listing 124:

```
88 uint256 maxIndex;
   uint256 maxCollateral;
```

3.125 CVF-125

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description This loop obtains the borrowing limits and the collateral factors for the same markets again and again on every iteration of the main loop. This is very inefficient.
Recommendation Consider obtaining these values for all markets before the main loop and storing into an in-memory array.

Listing 125:

```
90 for (uint256 i = 0; i < localMarkets.length; i++) {
```

3.126 CVF-126

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description The expression "localMarkets[i]" is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 126:

```
92 localMarkets[i] != address(0) &&
IMarket(localMarkets[i]).borrowingLimit(_borrower) > 0 &&
IMarket(localMarkets[i]).getCollateralFactor() > maxCollateral

96 maxCollateral = IMarket(localMarkets[i]).getCollateralFactor();
```

3.127 CVF-127

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description The borrowing limit is calculated twice for the market with the maximum collateral factor.
Recommendation Consider calculating once and reusing.

Listing 127:

```
93         IMarket(localMarkets[i]).borrowingLimit(_borrower) > 0
             ↪ &&

100 uint256 borrowingLimit = IMarket(localMarkets[maxIndex]).
    ↪ borrowingLimit(_borrower);
```

3.128 CVF-128

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description The expression "IMarket(localMarkets[i]).getCollateralFactor()" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 128:

```
94 IMarket(localMarkets[i]).getCollateralFactor() > maxCollateral

96 maxCollateral = IMarket(localMarkets[i]).getCollateralFactor();
```

3.129 CVF-129

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Comptroller.sol

Description The value "markets[maxIndex]" is equivalent to the already calculated value "localMarkets[maxIndex]".
Recommendation Consider reusing the already calculating value.

Listing 129:

```
100 uint256 borrowingLimit = IMarket(localMarkets[maxIndex]).
    ↪ borrowingLimit(_borrower);

106 IMarket(markets[maxIndex]).sendCollateralToLiquidator(
    ↪ _liquidator, _borrower, toPay);
```

3.130 CVF-130

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description As FACTOR / RATIOS is 100, this is equivalent to: uint256 collateral = borrowingLimit.mul(100).div(maxCollateral);

Listing 130:

```
101 uint256 collateralFactor = maxCollateral.mul(RATIOS);

103 uint256 collateral = borrowingLimit.mul(FACTOR).div(
    ↪ collateralFactor);

128 uint256 collateralFactor = IMarket(markets[i]).
    ↪ getCollateralFactor().mul(RATIOS);
    uint256 collateral = borrowingLimit.mul(FACTOR).div(
    ↪ collateralFactor);
```

3.131 CVF-131

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** Comptroller.sol

Description In case all the markets with non-zero borrowing limit are already processed, the 'maxIndex' and 'maxCollateral' variables will be zero. If the market with zero index wasn't yet processed (because its borrowing limit is zero), then it will be processed and zeroed out from the "localMarkets" array. Then, on the next iteration of the main loop, it will be "processed" again, but as the market address is already zeroed out, the whole transaction will be reverted.

Listing 131:

```
102 delete localMarkets[maxIndex];
```

3.132 CVF-132

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Comptroller.sol

Recommendation Plain (unsafe) addition could be used here as the value of 'marketsProcessed' is guaranteed to be less than 'localMarkets.length' here.

Listing 132:

```
107 marketsProcessed = marketsProcessed.add(1);
```

3.133 CVF-133

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Comptroller.sol

Description The expression "IMarket(markets[i]) is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 133:

```
126 uint256 borrowingLimit = IMarket(markets[i]).borrowingLimit(
    ↪ _borrower);

128 uint256 collateralFactor = IMarket(markets[i]).
    ↪ getCollateralFactor().mul(RATIOS);

132 IMarket(_markets[i]).sendCollateralToLiquidator(_liquidator,
    ↪ _borrower, toPay);
```

3.134 CVF-134

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Comptroller.sol

Recommendation The return type for this function should be "IMarket[] memory".

Listing 134:

```
138 function getAllMarkets() public view returns (address[] memory)
    ↪ {
```

3.135 CVF-135

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Comptroller.sol

Recommendation This function should emit some event and should probably be defined in the "IComptroller" interface.

Listing 135:

```
144 function resetMarkets() external onlyOwner {
```

3.136 CVF-136

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Comptroller.sol

Recommendation This function should emit some event and should probably be defined in the "IComptroller" interface.

Listing 136:

```
151 function removeMarket(address _market) external onlyOwner {
```

3.137 CVF-137

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Comptroller.sol

Recommendation The argument type should be "IMarket".

Listing 137:

```
151 function removeMarket(address _market) external onlyOwner {
```

3.138 CVF-138

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** Comptroller.sol

Description This function is inefficient as it rewrites the whole array in the storage.
Recommendation An efficient way to remove an element from an array is to: 1. Find the index of the element 2. In case the element is not the last one, overwrite the element to be removed with the last element of the array 3. Reduce the array length by one (do "pop()" on the array)

Listing 138:

```
151 function removeMarket(address _market) external onlyOwner {
```

3.139 CVF-139

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** PriceProxy.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 139:

```
6 constructor(address _logic, address _proxyAdmin) public
    ↪ TransparentUpgradeableProxy(_logic, _proxyAdmin, "") {}
```

3.140 CVF-140

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** MarketProxy.sol

Recommendation The type of the "_logic" argument should be "IMarket".

Listing 140:

```
6 constructor(address _logic, address _proxyAdmin) public
    ↪ TransparentUpgradeableProxy(_logic, _proxyAdmin, "") {}
```

3.141 CVF-141

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** MarketProxy.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 141:

```
6 constructor(address _logic, address _proxyAdmin) public
    ↪ TransparentUpgradeableProxy(_logic, _proxyAdmin, "") {}
```

3.142 CVF-142

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** LiquidityPoolProxy.sol

Recommendation The type of the "_logic" argument should be "ILiquidityPool".

Listing 142:

```
6 constructor(address _logic, address _proxyAdmin) public
    ↪ TransparentUpgradeableProxy(_logic, _proxyAdmin, "") {}
```

3.143 CVF-143

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** LiquidityPoolProxy.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 143:

```
6 constructor(address _logic, address _proxyAdmin) public
    ↪ TransparentUpgradeableProxy(_logic, _proxyAdmin, "") {}
```

3.144 CVF-144

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** ComptrollerProxy.sol

Recommendation The type of the "_logic" argument should be "IComptroller".

Listing 144:

```
6 constructor(address _logic, address _proxyAdmin) public
    ↪ TransparentUpgradeableProxy(_logic, _proxyAdmin, "") {}
```

3.145 CVF-145

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** ComptrollerProxy.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 145:

```
6 constructor(address _logic, address _proxyAdmin) public
    ↪ TransparentUpgradeableProxy(_logic, _proxyAdmin, "") {}
```

3.146 CVF-146

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** IXU3LP.sol

Description The semantics of these functions and the number format of the returned values are unclear.
Recommendation Consider adding documentation comments.

Listing 146:

```
5 function getNav() external view returns (uint256);

7 function getAmountInAsset0Terms(uint256) external view returns (
    ↪ uint256);
```

3.147 CVF-147

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** IXKNC.sol

Description The semantics of this function and the number format of the returned value are unclear.
Recommendation Consider adding a documentation comment.

Listing 147:

```
5 function getFundKncBalanceTwei() external view returns (uint256)
    ↪ ;
```

3.148 CVF-148

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IMarket.sol

Recommendation Documentation comments for the functions defined in this interface should be moved from the implementing smart contract to the interface itself.

Listing 148:

```
4 interface IMarket {
```

3.149 CVF-149

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IMarket.sol

Recommendation These functions should emit some events and these events should be defined in this interface.

Listing 149:

```
7 function setCollateralFactor(uint256 _collateralFactor) external
    ↪ ;

11 function setCollateralCap(uint256 _collateralCap) external;

21 function setCollateralizationActive(bool _active) external;
```

3.150 CVF-150

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** IMarket.sol

Recommendation The argument type should be "IComptroller".

Listing 150:

```
19 function setComptroller(address _comptroller) external;
```

3.151 CVF-151

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** ILPT.sol

Recommendation Documentation comments for the functions defined in this interface should be moved from the implementing smart contract to the interface itself.

Listing 151:

```
4 interface ILPT {
```

3.152 CVF-152

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** ILPT.sol

Description Indicating errors by return value is discouraged.
Recommendation Consider reverting in case of error. Those, who need to handle errors, could always use a try/catch block.

Listing 152:

```
5 function mint(address _recipient, uint256 _amount) external
    ↪ returns (bool);

7 function burnFrom(address _sender, uint256 _amount) external
    ↪ returns (bool);
```

3.153 CVF-153

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** ILiquidityPool.sol

Recommendation Documentation comments for the functions defined in this interface should be moved from the implementing smart contract to the interface itself.

Listing 153:

```
4 interface ILiquidityPool {
```

3.154 CVF-154

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** ILiquidityPool.sol

Description The word "updated" in the function name is confusing, as the function just returns the current (up to date) debt of a borrower.

Listing 154:

```
5 function updatedBorrowBy(address _borrower) external view
    ↪ returns (uint256);
```