Lybra Finance

Scope

The code under review can be found within the C4 Lybra Finance repository.

Summary

Findings

ID	Issue	Severity
M-01	Incorrect external function calls prevent users from depositing asset to mint PeUSD	Medium
L-01	Last withdrawal time updated though there was no withdrawal	Low
L-02	Modifiers should only implement checks and not make state changes	Low
L-03	Return values not checked for PeUSD and EUSD functions	Low
L-04	Require check should be added to ensure safe typecast	Low
L-05	Division before multiplication leads to loss of precision	Low

ID	Issue	Severity
N-01	Missing event emission for critical state changes	Non-Critical
N-02	Incorrect contract address in comment	Non-Critical
N-03	Contracts should avoid copy pasting functions from OFTV2.sol and instead inherit from it	Non-Critical
N-04	A pool setter function should be implemented for upgradeable token contracts	Non-Critical
N-05	increaseAllowance() should use an unchecked block pattern similar to decreaeAllowance()	Non-Critical

Analysis Report

Findings

[M-01] Incorrect external function calls prevent users from depositing asset to mint PeUSD

There are 2 instances of this issue:

When depositEtherToMint() function calls getAssetPrice(), it reverts since an invalid external call is made. This prevents a user from depositing ether to mint PeUSD. This can be a form of DOS.
https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/pools/LybraRETHVault.sol#L47
Correct function in RETH contract is getExchangeRate() not getExchangeRatio()

```
File: contracts/lybra/pools/LybraRETHVault.sol
46: function getAssetPrice() public override returns (uint256) {
47:         return (_etherPrice() *
IRETH(address(collateralAsset)).getExchangeRatio()) / 1e18;
48:     }
```

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/pools/LybraWbETHVault.sol#L35 Correct function in WBETH contract is exchangeRate() and not exchangeRatio().

```
File: contracts/lybra/pools/LybraWbETHVault.sol
34:  function getAssetPrice() public override returns (uint256) {
35:          return (_etherPrice() *
IWBETH(address(collateralAsset)).exchangeRatio()) / 1e18;
36:      }
```

[L-01] Last withdrawal time updated though there was no withdrawal

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/miner/ProtocolRewardsPool.sol#L105

If amount > 0 condition returns false, the if block is not executed, which means no LBR withdrawal occurred. But Line 105 still updates the lastWithdrawTime.

```
File: contracts/lybra/miner/ProtocolRewardsPool.sol
100: function withdraw(address user) public {
101:          uint256 amount = getClaimAbleLBR(user);
102:          if (amount > 0) {
103:              LBR.mint(user, amount);
104:          }
105:          lastWithdrawTime[user] = block.timestamp;
106:          emit WithdrawLBR(user, amount, block.timestamp);
107:      }
```

[L-02] Modifiers should only implement checks and not make state changes

There are 3 instances of this issue, where modifier updateReward() makes state changes to the contract:

1. stakerewardv2pool.sol
2. ProtocolRewardsPool.sol
3. EUSDMintingIncentives.sol

Make updateReward() into an internal function instead of a modifier.

[L-03] Return values not checked for PeUSD and EUSD functions

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/pools/base/LybraPeUSDVaultBase.sol#L181 There are 4 instances of this in LybraPeUSDVaultBase.sol:

```
File: contracts/lybra/pools/base/LybraPeUSDVaultBase.sol
181: PeUSD.mint(_onBehalfOf, _mintAmount);
199: PeUSD.transferFrom(_provider, address(configurator), totalFee);
200: PeUSD.burn(_provider, amount - totalFee);
203: PeUSD.transferFrom(_provider, address(configurator), amount);
```

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/pools/base/LybraEUSDVaultBase.sol#L264 There are 2 instances of this in LybraEUSDVaultBase.sol

```
File: contracts/lybra/pools/base/LybraEUSDVaultBase.sol
264: EUSD.mint(_onBehalfOf, _mintAmount);
279: EUSD.burn(_provider, amount);
```

[L-04] Require check should be added to ensure safe typecast

For all instances below, if a require check is not added and either the etherPrice or lbrPrice comes to be negative (highly unlikely), the typecast from int to uint will result in an underflow. https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/miner/EUSDMiningIncentives.sol#L153 Instead of this:

```
File: contracts/lybra/miner/EUSDMiningIncentives.sol
151: (, int etherPrice, , , ) = etherPriceFeed.latestRoundData();
152: (, int lbrPrice, , , ) = lbrPriceFeed.latestRoundData();
153: uint256 etherInLp =
(IEUSD(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2).balanceOf(ethlbrLpToken
) * uint(etherPrice)) / 1e8;
154: uint256 lbrInLp = (IEUSD(LBR).balanceOf(ethlbrLpToken) *
uint(lbrPrice)) / 1e8;
```

Use this:

```
File: contracts/lybra/miner/EUSDMiningIncentives.sol
151: (, int etherPrice, , , ) = etherPriceFeed.latestRoundData();
152: (, int lbrPrice, , , ) = lbrPriceFeed.latestRoundData();
153: require(etherPrice >= 0, "etherPrice is below zero");
154: require(lbrPrice >= 0, "lbrPrice is below zero");
155: uint256 etherInLp =
(IEUSD(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2).balanceOf(ethlbrLpToken
) * uint(etherPrice)) / 1e8;
156: uint256 lbrInLp = (IEUSD(LBR).balanceOf(ethlbrLpToken) *
uint(lbrPrice)) / 1e8;
```

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/miner/EUSDMiningIncentives.sol#L212 Instead of this:

```
File: contracts/lybra/miner/EUSDMiningIncentives.sol
212: (, int lbrPrice, , , ) = lbrPriceFeed.latestRoundData();
213: biddingFee = biddingFee * uint256(lbrPrice) / 1e8;
```

Use this:

```
File: contracts/lybra/miner/EUSDMiningIncentives.sol
212: (, int lbrPrice, , , ) = lbrPriceFeed.latestRoundData();
213: require(lbrPrice >= 0, "lbrPrice is below zero");
214: biddingFee = biddingFee * uint256(lbrPrice) / 1e8;
```

These require checks ensure that there cannot be an underflow if price is negative.

[L-05] Division before multiplication leads to loss of precision

Performing multiplication before division is generally better to avoid loss of precision because Solidity integer division might truncate.

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/pools/base/LybraEUSDVaultBase.sol#L239

```
File: contracts/lybra/pools/base/LybraEUSDVaultBase.sol
239: uint256 collateralAmount = (((eusdAmount * 1e18) / assetPrice) *
(10000 - configurator.redemptionFee())) / 10000;
```

[N-01] Missing event emission for critical state changes

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/miner/EUSDMiningIncentives.sol#L84 All functions from Line 84-130 do not have events and their emissions even though they change state variables.

```
File: contracts/lybra/miner/EUSDMiningIncentives.sol
84:     function setToken(address _lbr, address _eslbr) external onlyOwner
{
85:         LBR = _lbr;
86:         esLBR = _eslbr;
87:     }
88:
```

```
89:     function setLBROracle(address _lbrOracle) external onlyOwner {
90:         lbrPriceFeed = AggregatorV3Interface(_lbrOracle);
91:     }
92:
93:     function setPools(address[] memory _pools) external onlyOwner {
94:         for (uint i = 0; i < _pools.length; i++) {
95:             require(configurator.mintVault(_pools[i]), "NOT_VAULT");
96:         }
97:         pools = _pools;
98:     }
99:
100:     function setBiddingCost(uint256 _biddingRatio) external onlyOwner
{
101:         require(_biddingRatio <= 8000, "BCE");
102:         biddingFeeRatio = _biddingRatio;
103:     }
104:
105:     function setExtraRatio(uint256 ratio) external onlyOwner {
106:         require(ratio <= 1e20, "BCE");
107:         extraRatio = ratio;
108:     }
109:
110:     function setPeUSDExtraRatio(uint256 ratio) external onlyOwner {
111:         require(ratio <= 1e20, "BCE");
112:         peUSDExtraRatio = ratio;
113:     }
114:
115:     function setBoost(address _boost) external onlyOwner {
116:         esLBRBoost = IesLBRBoost(_boost);
117:     }
118:
119:     function setRewardsDuration(uint256 _duration) external onlyOwner
{
120:         require(finishAt < block.timestamp, "reward duration not
finished");
121:         duration = _duration;
122:     }
123:
124:     function setEthlbrStakeInfo(address _pool, address _lp) external
onlyOwner {
125:         ethlbrStakePool = _pool;
126:         ethlbrLpToken = _lp;
127:     }
128:     function setEUSDBuyoutAllowed(bool _bool) external onlyOwner {
129:         isEUSDBuyoutAllowed = _bool;
130:     }
```

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/configuration/LybraConfigurator.sol#L98 Event emission missing for these three functions.

```
File: contracts/lybra/configuration/LybraConfigurator.sol
98: function initToken(address _eusd, address _peusd) external
onlyRole(DAO) {
99:         if (address(EUSD) == address(0)) EUSD = IEUSD(_eusd);
100:         if (address(peUSD) == address(0)) peUSD = IEUSD(_peusd);
101:     }

109: function setMintVault(address pool, bool isActive) external
onlyRole(DAO) {
110:         mintVault[pool] = isActive;
111:     }

119: function setMintVaultMaxSupply(address pool, uint256 maxSupply)
external onlyRole(DAO) {
120:         mintVaultMaxSupply[pool] = maxSupply;
121:     }
```

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/configuration/LybraConfigurator.sol#L158C5-L178C42 Event emission missing for these three functions.

```
File: contracts/lybra/configuration/LybraConfigurator.sol
158:     function setvaultBurnPaused(address pool, bool isActive) external
checkRole(TIMELOCK) {
159:         vaultBurnPaused[pool] = isActive;
160:     }

167:     function setPremiumTradingEnabled(bool isActive) external
checkRole(TIMELOCK) {
168:         premiumTradingEnabled = isActive;
169:     }

177:    function setvaultMintPaused(address pool, bool isActive) external
checkRole(ADMIN) {
178:         vaultMintPaused[pool] = isActive;
179:     }
```

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/configuration/LybraConfigurator.sol#L246 Event emission missing for these two functions. Note: function on Line 253 already has an event emission but it should be placed after Line 256.

```
File: contracts/lybra/configuration/LybraConfigurator.sol
246:     function setMaxStableRatio(uint256 _ratio) external
checkRole(TIMELOCK) {
247:         require(_ratio <= 10_000, "The maximum value is 10000");
248:         maxStableRatio = _ratio;
249:     }
```

```
253:     function setFlashloanFee(uint256 fee) external
checkRole(TIMELOCK) {
254:         if (fee > 10_000) revert('EL');
255:         emit FlashloanFeeUpdated(fee);
256:         flashloanFee = fee;
257:     }

261:     function setProtocolRewardsToken(address _token) external
checkRole(TIMELOCK) {
262:         stableToken = _token;
263:     }
```

[N-02] Incorrect contract address in comment

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/pools/LybraWbETHVault.sol#L16 The contract address is of the rETH contract and not WBETH.

```
File: contracts/lybra/pools/LybraWbETHVault.sol
16: //WBETH = 0xae78736Cd615f374D3085123A210448E74Fc6393
```

[N-03] Contracts should avoid copy pasting functions from OFTV2.sol and instead inherit from it

The contracts below copy paste functions from the OFTV2.sol contract. It is a better design choice to inherit from an already existing contract (OFTV2.sol) and pass in the values to its constructor directly. Additionally, we do not need to import the BaseOFTV2.sol contract as well since the OFTV2.sol already inherits it.

LBR.sol PeUSD.sol PeUSDMainnetStableVision.sol

For example, the content of the peUSD contract can now only have:

```
contract PeUSD is OFTV2 {

constructor(uint8 _sharedDecimals, address _lzEndpoint) OFTV2("peg-eUSD",
"peUSD", _sharedDecimals,
_lzEndpoint) {
}
}
```

[N-04] A pool setter function should be implemented for upgradeable token contracts

If token contracts are upgraded to another address, it might make our contracts incompatible. To prevent this from happening, pool setter functions should be implemented in case a contract is upgraded in the future (like it is implemented in LybraRETHVault.sol)

Upgradeable contracts: WBETH.sol stETH.sol

Contracts requiring setter function: LybraWbETHVault.sol LybraWstETHVault.sol LybraStETHVault.sol

[N-05] increaseAllowance() should use an unchecked block pattern similar to decreaeAllowance()

https://github.com/code-423n4/2023-06-lybra/blob/7b73ef2fbb542b569e182d9abf79be643ca883ee/contracts/lybra/token/EUSD.sol#L248 It is better for functions with opposite functionalities to use similar patterns in it's code.

Instead of this:

```
File: contracts/lybra/token/EUSD.sol
248: function increaseAllowance(address _spender, uint256 _addedValue)
public returns (bool) {
249:         address owner = _msgSender();
250:         _approve(owner, _spender, allowances[owner]
[_spender].add(_addedValue));
251:         return true;
252:     }
```

Use this:

```
File: contracts/lybra/token/EUSD.sol
248: function inreaseAllowance(address spender, uint256 _addedValue)
public returns (bool) {
249:         address owner = _msgSender();
250:         uint256 currentAllowance = allowance(owner, spender);
251:         require(currentAllowance + _addedValue <= type(uint256).max,
"ERC20: increases allowance above max limit");
252:         unchecked {
253:             _approve(owner, spender, currentAllowance + _addedValue);
254:         }
255:
256:         return true;
257:     }
```

Analysis Report

Approach

1. Read Scope description of each contract on the C4 page to get a general overview on the architecture of the protocol.

2. After that, I delved into the contract code, starting from non-rebase vaults to rebase vaults i.e. LybraPeUSDVaultBase.sol and its child contracts to LybraEUSDVaultBase.sol and its child contracts.
3. The remaining contracts fall into 4 categories: Timelock contracts, Governance contracts, Reward contracts and Token contracts.
4. The LybraConfigurator.sol contract is a standalone contract, which is controlled by the DAO. This contract sets the various parameters and control functionalities of the Lybra Protocol.

This was the order in which I audited the contracts. In places where I could not understand a certain functionality, I dmed the sponsors and referred to the documentation to seek additional context.

Architecture Recommendations

The protocol is designed in an easy to understand way. There are certain contracts that can improve the way they are designed. For example, in one of my findings, I've reported three (LBR, PeUSD, PeUSDMainnetStableVision) token contracts that could directly inherit from the LayerZero OFTV2.sol contract instead of copy pasting its functions (keeping the endpoints simple through which cross-chain communication occurs is recommended). Another one where a setter function was not implemented for upgradeable token contracts in specific LST vaults. These minute architecture recommendations have been included in the QA report I submitted. Personally, architecture-wise I would rate it on a Medium level.

What is unique in the architecture?

The use of the LayerZero OFT contracts, which enable cross chain communication is unique. I did not delve deeper on the OFT contracts side but just checked the return values and some of the internal functions utilized.

Codebase quality analysis

With the absence of NatSpec comments and a fully complete documentation, it was difficult to understand the assumptions being made on the developer side. For example, how stake, unstake and withdraw mechanism would work. The errors related to staking and unstaking have been submitted as HMs. Other than that, there were several low issues and non-critical ones I've included. These include the inconsistent patterns used in the codebase such as inconsistent event emissions, using unchecked in decreaseAllowance but not in increaseAllowance and the require checks being used in depositAssetAmount() being different in both LybraPeUSDVaultBase.sol and LybraEUSDVaultBase.sol contracts. Overall, from my point of view the codebase was not ready for an audit due to the presence of barely any tests, absence of in-depth documentation and typo errors in external function calls.

Centralization risks

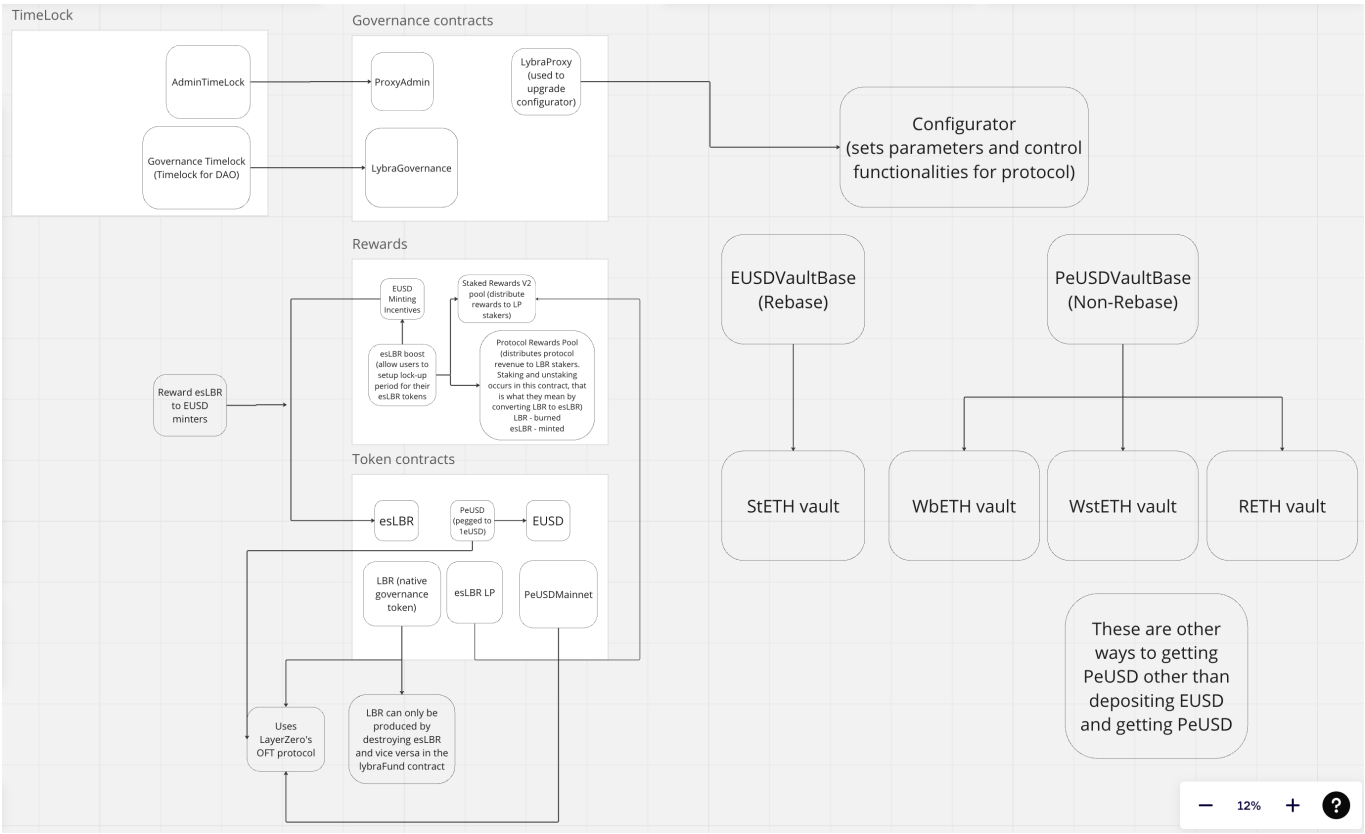
Some degree of centralization is introduced through the Governance contracts but the presence of timelock contracts on them tilt it towards some degree of decentralisation. Mainly the access control exists in the LybraConfigurator, which I think does not pose much risk. I think this is a common pattern used in most protocols. On the other hand, the owner in the reward contracts poses a centralization risk since it has control over most functions which could destabilize the system. Additionally not using a two-step change for critical addresses like the owner can lead to the contract ownership being transferred to an unwanted address.

Systemic risks

From the issues I've found the only systemic risks existing are those in the ProtocolRewardsPool.sol contract and PeUSDMainnetStableVision.sol. In the former, one can tamper with the stake/unstake functionality while in the latter the system is at risk due to the absence of rate limiting when borrowing a flash loan. There are several forms of DOS existing in the contract logic itself, which prevent users from utilizing a service in the protocol (included in findings). In addition to that, the centralization risks above pose a systemic risk as well.

Mechanical Review

Here is a mechanical mindmap to understand the architecture and some interactions of the protocol: https://drive.google.com/file/d/1EwQTqNQWdRx1zAMkVbvmGg1ySGncJwRe/view?usp=sharing I have provided edit access to draw on top of the image if necessary.



New insights and learnings from this audit

1. This was my first time auditing a LSD-based vault. Auditing this protocol has helped me in understanding how LSDs work on a deeper level. For example, business logic for excess income distribution, redemption time for staking, and how token boosts work.
2. Auditing without tests is a bit difficult. Learning how to write tests in foundry through Patrick Collins' YT is something I am focusing on right now after this audit is over. It surely puts you one step ahead when writing POCs and finding bugs.
3. There are some OZ libraries in this protocol that I had not delved into before. Understanding how they work while auditing the protocol provided some insight on how it fits in.
4. Lastly, this is the first time I've seen LayerZero contracts being used in a protocol. Understanding the endpoints from which cross-chain communication occurs in this project is something new I learned from an architectural standpoint.

Time spent:

70 hours