Competitive Security Assessment

Lends-Protocol

Sep 30th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.
The comprehensive examination and auditing scope includes:
• Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
• Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
• Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
• Verify the code base is compliant with the most up-to-date industry standards and security best practices.
• Comprehensive line-by-line manual code review of the entire codebase by industry experts.
The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

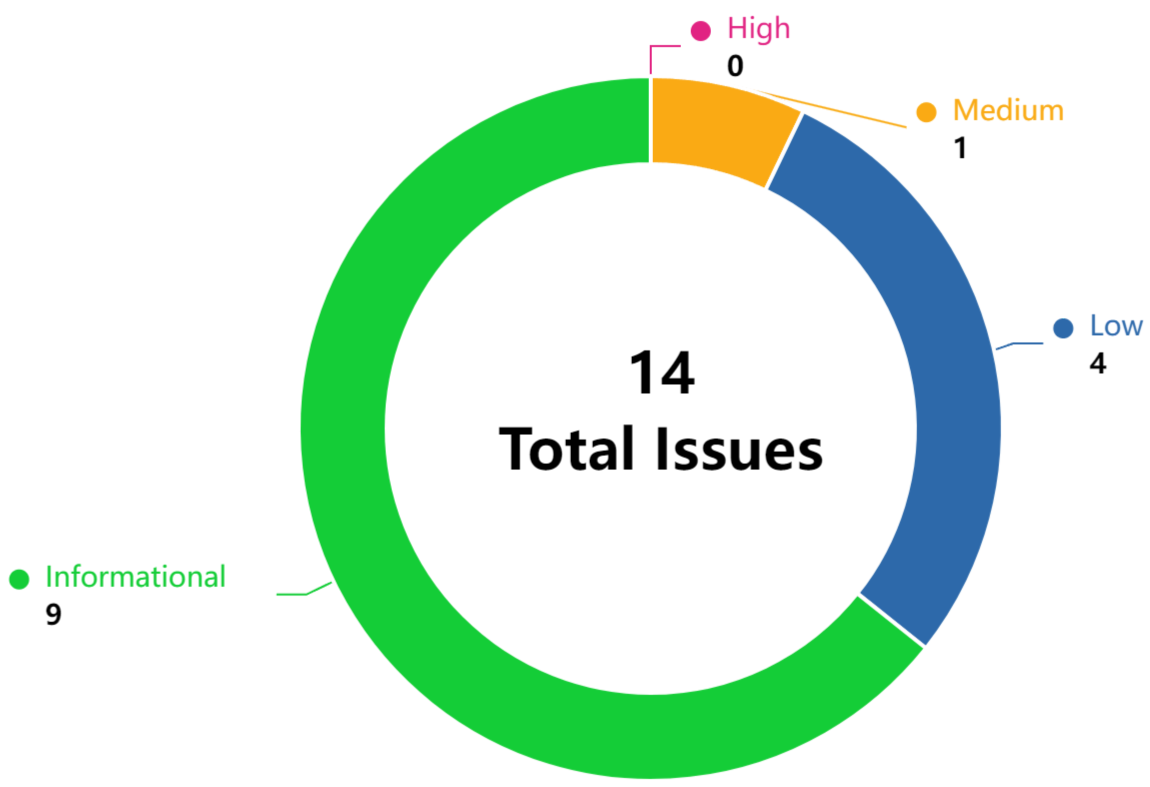
Overview

Project Name	Lends-Protocol
Language	solidity
Codebase	• https://github.com/lends-so/lends-p2p-lending • audit version-e4846724028a4d8b16ae6b63d9f4b787a5872425 • final version-ac521d10a6bc207bfa9926877282e2b6653bd4fb

Audit Scope

File	SHA256 Hash
contracts/Pool.sol	562c587b3a22875bc98f6c47d3b48d72fe88bd6e573132d30c7123eec2ad6fdd
contracts/PoolFactory.sol	a168cfb0f7dd07b0b821f55a92f3b3a315eef63980a0c76ebfc6b2ac0cf500ed
contracts/Oracle.sol	49db783069140778875ac6f1334d4b702c12f32d24a76d56453e0b37510fcc1b

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
LDP-1	No way to update `Oracle` for existing pools	Logical	Medium	Fixed	***
LDP-2	Unrestricted Expiry Extension in Pool Contract	Logical	Low	Fixed	***
LDP-3	Missing validation for `addWhitelist`	Logical	Low	Fixed	***
LDP-4	L2 Sequencer is down will compromise oracle price feed	Logical	Low	Fixed	***
LDP-5	Incompatibility With Fee-on-Transfer Tokens	Logical	Low	Declined	***
LDP-6	Denominator too high may cause the final value go to zero	Logical	Low	Fixed	***
LDP-7	unchecked for loop	Gas Optimization	Informational	Fixed	***

LDP-8	repayLoan() function is not borrower friendly since it does not allow for partial repayment.	Logical	Informational	Acknowledged	***
LDP-9	Unnecessary Direct Manipulation of State Variable	Language Specific	Informational	Fixed	***
LDP-10	Two-step ownership transfer	Logical	Informational	Fixed	***
LDP-11	Token-approved Gas Optimization Proposals	Gas Optimization	Informational	Fixed	***
LDP-12	Redundant code	Code Style	Informational	Fixed	***
LDP-13	Packed variables for `Pool` contract	Gas Optimization	Informational	Fixed	***
LDP-14	Missing validation for `allBorrowers`	Logical	Informational	Fixed	***
LDP-15	Insufficient Input Validation in Oracle Contract Constructor	Logical	Informational	Fixed	***

LDP-1:No way to update `Oracle` for existing pools

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	***

Code Reference

- code/contracts/Pool.sol#L78

```
78: oracle = IOracle(_oracle);
```

- code/contracts/PoolFactory.sol#L92-L97

```
92: function setOracle(address _oracle) external onlyOwner {
93:         if (_oracle == address(0)) revert ZeroAddress();
94: 
95:         oracle = _oracle;
96:         emit OracleAddressUpdated(oracle);
97:     }
```

Description

*****: In the protocol, users can deploy pool contracts using the deployPool function of the PoolFactory contract, with the oracle from the PoolFactory being passed as an initialization parameter for the pool deployment. However, it's important to note that the oracle parameter value of the pool is fixed and cannot be modified once the pool contract is deployed, while the oracle parameter value in the PoolFactory contract can be changed using the setOracle function. This means that if the old oracle becomes outdated or poses security risks, the PoolFactory contract can modify it, but the pool contract is powerless to do so, which poses a significant risk to the stability of the protocol.

Recommendation

*****: It is recommended that when the Pool contract uses an oracle, it should directly call the oracle interface of the PoolFactory to obtain the value, rather than treating the oracle as a fixed value.
Consider the following fixes:

```
function getLoanAmount(
    uint256 _amount
) public view returns (uint256 loanAmount) {
    IOracle oracle = poolFactory.oracle();
    int256 collateralPrice = oracle.getPriceUSD(collateralToken);
    int256 poolTokenPrice = oracle.getPriceUSD(poolToken);
    ...
}
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@004b568

LDP-2:Unrestricted Expiry Extension in Pool Contract

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	***

Code Reference

- code/contracts/Pool.sol#L368-L373

```
368: function extendExpiry(uint256 _expiry) external onlyOwner {
369:         if (_expiry <= block.timestamp) revert InvalidExpiry();
370:         if (_expiry <= expiry) revert CanOnlyExtendExpiry();
371:         expiry = _expiry;
372:         emit ExtendedExpiry(_expiry);
373:     }
```

Description

***: The Pool contract's `extendExpiry` function lacks a crucial time-based restriction, allowing the owner to extend the pool's expiry even after its intended lifecycle has ended. This oversight can lead to unexpected contract state changes and potential manipulation of the pool's operational timeline.
Current implementation:

```
function extendExpiry(uint256 _expiry) external onlyOwner {
    if (_expiry <= block.timestamp) revert InvalidExpiry();
    if (_expiry <= expiry) revert CanOnlyExtendExpiry();
    expiry = _expiry;
    emit ExtendedExpiry(_expiry);
}
```

This implementation allows the owner to extend the expiry at any time, even after the pool has expired and potentially after the `withdraw` function has been called.
Impact:

1. Violation of the expected pool lifecycle, potentially disrupting user expectations and strategies.
2. Possibility of reactivating an expired pool, leading to inconsistent contract states.
3. Potential for abuse by allowing the owner to manipulate the pool's operational period arbitrarily.

Proof of Concept:

1. Pool is created with an expiry of T+30 days.
2. At T+31 days, the pool has expired, and `withdraw` function is called.
3. At T+32 days, owner calls `extendExpiry`, setting a new expiry at T+60 days.
4. The pool is now unexpectedly active again, potentially after funds have been withdrawn.

Recommendation

***: Modify the `extendExpiry` function to include a time-based restriction:

```
function extendExpiry(uint256 _expiry) external onlyOwner beforeExpiry {
    if (_expiry <= block.timestamp) revert InvalidExpiry();
    if (_expiry <= expiry) revert CanOnlyExtendExpiry();
    expiry = _expiry;
    emit ExtendedExpiry(_expiry);
}
```

This modification:

1. Ensures that expiry can only be extended before the current expiry date.
2. Prevents manipulation of the pool's lifecycle after it has ended.

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@63c115d

LDP-3:Missing validation for `addWhitelist`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	***

Code Reference

- code/contracts/Pool.sol#L334-L338

```
334: function addWhitelist(address _address) external onlyOwner {
335:         if (_address == address(0)) revert ZeroAddress();
336:         whitelist.push(_address);
337:         emit AddedWhiteList(_address);
338:     }
```

Description

***: There is no validation to check if `_address` has already been added to `whitelist`, so the same address can be added multiple times. If so, when user has been removed from `whitelist`, he is still in the white list.

Recommendation

***: Add validation to prevent the address from being added multiple times to white list.

```
    function addWhitelist(address _address) external onlyOwner {
        if (_address == address(0)) revert ZeroAddress();
        uint256 length = whitelist.length;
        bool isExists = false;
        for (uint256 i = 0; i < length;) {
            if (whitelist[i] == _address) {
                isExists = true;
                break;
            }
            unchecked{
                i++;
            }
        }
        if(!isExists){
            whitelist.push(_address);
        }
        emit AddedWhiteList(_address);
    }
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@00f6771

LDP-4:L2 Sequencer is down will compromise oracle price feed

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	***

Code Reference

- code/contracts/Oracle.sol#L83-L97

```
function _getLatestRoundData(
        address _feed
    ) internal view returns (int256 latestPrice) {
        AggregatorV3Interface feed = AggregatorV3Interface(_feed);
        (
            uint80 roundId,
            int256 price,
            ,
            /* uint256 startedAt */
            uint256 timestamp,
            uint80 answeredInRound
        ) = feed.latestRoundData();
        if (!_isValidRound(roundId, timestamp, answeredInRound)) return 0;
        return price;
    }
```

Description

*****:** The Lends.so website shows that it will be deployed at L2. In most L2 protocols, the sequencer is primarily responsible for rolling up L2 transactions. When the sequencer is down, transactions and data on L2 cannot be processed correctly. Therefore, when using the Chainlink oracle, it is necessary to check whether the current L2 sequencer status is available. Chainlink provides sequencerUptimeFeed to check the availability of the current L2 sequencer status.
Ref: https://docs.chain.link/data-feeds/l2-sequencer-feeds

Recommendation

*****:** It is recommended to check whether the sequencer is available when using Chainlink to obtain prices in L2.
Consider the following fixes:

```solidity
uint256 private constant GRACE_PERIOD_TIME = 3600;

function _getLatestRoundData(
    address _feed
) internal view returns (int256 latestPrice) {
    ...
    (
        /*uint80 roundID*/,
        int256 answer,
        uint256 startedAt,
        /*uint256 updatedAt*/,
        /*uint80 answeredInRound*/
    ) = sequencerUptimeFeed.latestRoundData();

    // Answer == 0: Sequencer is up
    // Answer == 1: Sequencer is down
    bool isSequencerUp = answer == 0;
    if (!isSequencerUp) {
        revert SequencerDown();
    }

    // Make sure the grace period has passed after the
    // sequencer is back up.
    uint256 timeSinceUp = block.timestamp - startedAt;
    if (timeSinceUp <= GRACE_PERIOD_TIME) {
        revert GracePeriodNotOver();
    }
    ...
    AggregatorV3Interface feed = AggregatorV3Interface(_feed);
    (
        uint80 roundId,
        int256 price,
        ,
        /* uint256 startedAt */
        uint256 timestamp,
        uint80 answeredInRound
    ) = feed.latestRoundData();
    if (!_isValidRound(roundId, timestamp, answeredInRound)) return 0;
    return price;
}
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@0be3392

LDP-5:Incompatibility With Fee-on-Transfer Tokens

Category	Severity	Client Response	Contributor
Logical	Low	Declined	***

Code Reference

- code/contracts/Pool.sol#L384-L392

```
384: function _safeTransferFrom(
385:         address _token,
386:         address _from,
387:         address _to,
388:         uint256 _amount
389:     ) internal {
390:         bool success = IERC20(_token).transferFrom(_from, _to, _amount);
391:         if (!success) revert TransferFailed();
392:     }
```

- code/contracts/PoolFactory.sol#L55-L59

```
55: bool success = IERC20(params.poolToken).transferFrom(
56:             msg.sender,
57:             address(this),
58:             params.poolDeposit
59:         );
```

Description

***: The function `transferFrom` will transfer the poolToken to the contract:

```
bool success = IERC20(params.poolToken).transferFrom(
            msg.sender,
            address(this),
            params.poolDeposit
        );
```

We can find that the balance of `msg.sender` will record the input amount `params.poolDeposit`, however, the issue is here that if the `poolToken` is a FoT token. Certain tokens (e.g., STA or PAXG) charge a fee for transfers and others (e.g., USDT or USDC) may start doing so in the future. As a result, the actual amount received of tokens will be less than input amount, due to the charged transaction fee.
You can find the critical hack events in several `Balancer` pools because of FoT token vulnerability (more details).

Recommendation

***: Consider recording the actual amount received in the function:

```
uint256 balanceBefore = IERC20(params.poolToken).balanceOf(address(*this*));

bool success = IERC20(params.poolToken).transferFrom(
            msg.sender,
            address(this),
            params.poolDeposit
        );

uint256 actualAmount = IERC20(params.poolToken).balanceOf(address(*this*)) - balanceBefore;
        //..
    }
}
```

Or, use blacklist to reject the FoT tokens.

Client Response

client response : Declined. This is intentional as it will cause problems with other functionality if there's a fee on transfer.

LDP-6:Denominator too high may cause the final value go to zero

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	***

Code Reference

- code/contracts/Pool.sol#L111-L123
- code/contracts/Pool.sol#L259-L276

```
111: uint256 loanAmount = getLoanAmount(_amount);
112:
113:         if (loanAmount > poolDeposit) revert InsufficientPoolDeposit();
114:
115:         poolBorrowed += loanAmount;
116:         collateralDeposit += _amount;
117:
118:         loans[msg.sender] = Loan({
119:             borrower: msg.sender,
120:             entryTimestamp: block.timestamp,
121:             collateralAmount: loans[msg.sender].collateralAmount + _amount,
122:             loanAmount: loans[msg.sender].loanAmount + loanAmount
123:         });
```

```
259: (_amount * uint256(collateralPrice) * ltv) /
260:                 (uint256(poolTokenPrice) *
261:                     10 ** (collateralTokenDecimals - poolTokenDecimals) *
262:                     MAX_BASIS_POINTS);
263:
264:         /// @dev (collateralAmount * collateralPrice * ltv * decimalDelta) / (poolTokenPrice * MAX_BASIS_POINTS)
265:         if (poolTokenDecimals > collateralTokenDecimals)
266:             return
267:                 (_amount *
268:                     uint256(collateralPrice) *
269:                     ltv *
270:                     10 ** (poolTokenDecimals - collateralTokenDecimals)) /
271:                 (uint256(poolTokenPrice) * MAX_BASIS_POINTS);
272:
273:         /// @dev (collateralAmount * collateralPrice * ltv) / (poolTokenPrice * MAX_BASIS_POINTS)
274:         return
275:             (_amount * uint256(collateralPrice) * ltv) /
276:             (uint256(poolTokenPrice) * MAX_BASIS_POINTS);
```

Description

***:

```
      uint8 collateralTokenDecimals = IERC20Metadata(collateralToken)
            .decimals();
        uint8 poolTokenDecimals = IERC20Metadata(poolToken).decimals();

        /// @dev (collateralAmount * collateralPrice * ltv) / (poolTokenPrice * decimalDelta * MAX_BASIS_POI
NTS)
        if (collateralTokenDecimals > poolTokenDecimals)
            return
                (_amount * uint256(collateralPrice) * ltv) /
                (uint256(poolTokenPrice) *
                    10 ** (collateralTokenDecimals - poolTokenDecimals) *
                    MAX_BASIS_POINTS);

        /// @dev (collateralAmount * collateralPrice * ltv * decimalDelta) / (poolTokenPrice * MAX_BASIS_POI
NTS)
        if (poolTokenDecimals > collateralTokenDecimals)
            return
                (_amount *
                    uint256(collateralPrice) *
                    ltv *
                    10 ** (poolTokenDecimals - collateralTokenDecimals)) /
                (uint256(poolTokenPrice) * MAX_BASIS_POINTS);

        /// @dev (collateralAmount * collateralPrice * ltv) / (poolTokenPrice * MAX_BASIS_POINTS)
        return
            (_amount * uint256(collateralPrice) * ltv) /
            (uint256(poolTokenPrice) * MAX_BASIS_POINTS);
    }
```

case 1&case 2:

```
(_amount * uint256(collateralPrice) * ltv) /
                (uint256(poolTokenPrice) *
                    10 ** (collateralTokenDecimals - poolTokenDecimals) *
                    MAX_BASIS_POINTS)
```

if the denominator is too high, bigger than the numerator,the return value will be zero .
Take an example:
the _amount here is very low, 1.the collateralPrice is 20u each,ltv here is 7000.poolTokenPrice is 10u each,the collateralTokenDecimals - poolTokenDecimals is eaqul to 1,.
i think the data above is proper, but the return value is 0.14,equal to 0 . the loanAmount will be 0 . so after the following logic ,the pool token return
to borrower will be zero
case 3:

```
return
            (_amount * uint256(collateralPrice) * ltv) /
            (uint256(poolTokenPrice) * MAX_BASIS_POINTS);
```

The price of poolToken is several times that of collateralToken,for example:
amount is 10,collateralPrice is 10usd each,ltv is 1500,pooltokenPrice is 200usd each ,MAX_BASIS_POINTS is 10000.
the result is 10*10*1500/(200*10000)=0,the collateralToken sending into contract will be 0.
the data above is proper,user want to use 10 tokenA to change tokenB,but user find his money was eaten by the damn contract!

Recommendation

***: revert if the loanAmount is zero

```
  uint256 loanAmount = getLoanAmount(_amount);

      + require(loanAmount!=0,"loanAmount can't be zero")
        if (loanAmount > poolDeposit) revert InsufficientPoolDeposit();

        poolBorrowed += loanAmount;
        collateralDeposit += _amount;

        loans[msg.sender] = Loan({
            borrower: msg.sender,
            entryTimestamp: block.timestamp,
            collateralAmount: loans[msg.sender].collateralAmount + _amount,
            loanAmount: loans[msg.sender].loanAmount + loanAmount
        });

        allBorrowers.push(msg.sender);

        _safeTransferFrom(collateralToken, msg.sender, address(this), _amount);
        _safeApprove(poolToken, address(this), loanAmount);
        _safeTransferFrom(poolToken, address(this), msg.sender, loanAmount);

        emit LoanTaken(msg.sender, loanAmount);
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@2c1edcc

LDP-7:unchecked for loop

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	***

Code Reference

- code/contracts/Oracle.sol#L16-L18

```
16: for (uint256 i = 0; i < _usdPairs.length; ++i) {
17:             usdTokenFeeds[_usdPairs[i].token] = _usdPairs[i].feed;
18:         }
```

- code/contracts/Pool.sol#L102-L107
- code/contracts/Pool.sol#L351-L357

```
102: for (uint256 i = 0; i < whitelistLength; i++) {
103:                 if (whitelist[i] == msg.sender) {
104:                     isWhitelisted = true;
105:                     break;
106:                 }
107:             }
```

```
351: for (uint256 i = 0; i < whitelistLength; i++) {
352:             if (localWhitelist[i] == _address) {
353:                 localWhitelist[i] = localWhitelist[whitelistLength - 1];
354:                 localWhitelist.pop();
355:                 break;
356:             }
357:         }
```

- code/contracts/PoolFactory.sol#L133-L136

```
133: for (uint256 i = 0; i < poolCount; i++) {
134:             IPool pool = IPool(_allPools[i]);
135:             allMetadata[i] = pool.getPoolMetadata();
136:         }
```

Description

***: In Solidity 0.8+, there's a default overflow check on unsigned integers. It's possible to uncheck this in for-loops and save some gas at each iteration, but at the cost of some code readability, as this uncheck cannot be made inline.
You can see this in it :
https://solodit.xyz/issues/g-38-increments-can-be-unchecked-code4rena-jpegd-jpegd-contest-git

Recommendation

***: The code would go from:

```
for (uint256 i = 0; i < poolCount; i++) {
            // ...
}
```

to:

```
for (uint256 i = 0; i < poolCount;) {
            // ...
            unchecked { ++i; }
}
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@8e26423

LDP-8:repayLoan() function is not borrower friendly since it does not allow for partial repayment.

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	***

Code Reference

- code/contracts/Pool.sol#L205-L234

```
205: function repayLoan() external nonReentrant beforeExpiry {
206:         Loan storage loan = loans[msg.sender];
207:
208:         if (loan.collateralAmount == 0) revert InvalidPoolDeposit();
209:
210:         uint256 totalRepayment = getRepaymentAmount(
211:             loan.loanAmount,
212:             loan.entryTimestamp
213:         );
214:
215:         uint256 collateralAmount = loan.collateralAmount;
216:
217:         poolBorrowed -= loan.loanAmount;
218:         collateralDeposit -= collateralAmount;
219:
220:         loan.collateralAmount = 0;
221:         loan.loanAmount = 0;
222:         loan.entryTimestamp = 0;
223:
224:         _safeTransferFrom(poolToken, msg.sender, address(this), totalRepayment);
```

```
225:         _safeApprove(collateralToken, address(this), collateralAmount);
226:         _safeTransferFrom(
227:             collateralToken,
228:             address(this),
229:             msg.sender,
230:             collateralAmount
231:         );
232:
233:         emit LoanRepaid(msg.sender, totalRepayment);
234:     }
```

Description

***: From contest page which is derived from white paper.
Lends' P2P Lending platform offers a customizable, secure, and flexible way for lenders and borrowers to interact. Lenders can create their own lending pools by setting terms and seeding capital, while borrowers can secure loans by contributing collateral. `The platform emphasizes borrower-friendly features, including flexible repayment options that allow full, partial, or no repayment, with corresponding collateral management.` This approach ensures a balanced and user-controlled lending experience.
We can see that the `repayLoan` function allows to repay the loan fully. It does not allow the partial repayment.

Impact

The current implementation does not provide the option to borrower to repay the loan partially.

Recommendation

*****:** We would suggest to implement function to repay the loan partially.

Client Response

client response : Acknowledged. It's going to be a feature later on.

LDP-9:Unnecessary Direct Manipulation of State Variable

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	***

Code Reference

- code/contracts/Pool.sol#L345

```
345: function removeWhitelist(address _address) external onlyOwner {
```

Description

***: The `removeWhitelist` function creates an unnecessary storage reference (`localWhitelist`) to the whitelist state variable. This reference is redundant, as operations can be directly performed on whitelist. Additionally, the variable name `localWhitelist` is misleading, suggesting it is a local copy rather than a reference.

Recommendation

***: Use a cached memory version of the whitelist array to perform the necessary operations, then update the state variable after all changes are made. This can reduce gas costs and improve performance.
Revised Implementation:

```
function removeWhitelist(address _address) external onlyOwner {
    if (_address == address(0)) revert ZeroAddress();
    uint256 whitelistLength = whitelist.length;
    for (uint256 i = 0; i < whitelistLength; i++) {
        if (whitelist[i] == _address) {
            whitelist[i] = whitelist[whitelistLength - 1];
            whitelist.pop();
            break;
        }
    }

    emit RemovedWhiteList(_address);
}
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@fae2e3f

LDP-10:Two-step ownership transfer

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/Pool.sol#L66

```
66: ) Ownable(_owner) {
```

Description

***: Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. If the admin permissions are given to the wrong address within this function, it will cause irreparable damage to the contract.
Below is the official documentation explanation from OpenZeppelin:
https://docs.openzeppelin.com/contracts/4.x/api/access
Ownable is a simpler mechanism with a single owner "role" that can be assigned to a single account. This simpler mechanism can be useful for quick tests but projects with production concerns are likely to outgrow it.
The SanctionsList contract inherits from the Ownable contract, where the ownership transfer is a single-step process, posing this risk.

```
constructor(
        address _owner,
        address _factory,
        address _oracle,
        address _poolToken,
        address _collateralToken,
        uint256 _apr,
        uint256 _ltv,
        uint256 _expiry,
        address[] memory _whitelist
    ) Ownable(_owner) {
```

Recommendation

***: It is recommended to use the Ownable2Step contract from OZ (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol) instead.

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@8e26423

LDP-11:Token-approved Gas Optimization Proposals

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	***

Code Reference

- code/contracts/Pool.sol#L156-L201

```
156: _safeApprove(poolToken, address(this), fee);
157:             _safeTransferFrom(
158:                 poolToken,
159:                 address(this),
160:                 poolFactory.getFeeCollector(),
161:                 fee
162:             );
163:         }
164:
165:         poolDeposit = 0;
166:         collateralDeposit = 0;
167:         poolBorrowed = 0;
168:
169:         /// @dev If the borrower has defaulted on the loan
170:         if (collateralBalance > 0) {
171:             uint256 defaultedFee = (collateralBalance *
172:                 poolFactory.getFeeBasisPoints()) / MAX_BASIS_POINTS;
173:
174:             _safeApprove(collateralToken, address(this), defaultedFee);
175:             _safeTransferFrom(
176:                 collateralToken,
177:                 address(this),
178:                 poolFactory.getFeeCollector(),
179:                 defaultedFee
180:             );
181:
182:             _safeApprove(
183:                 collateralToken,
184:                 address(this),
185:                 collateralBalance - defaultedFee
186:             );
187:             _safeTransferFrom(
188:                 collateralToken,
189:                 address(this),
190:                 msg.sender,
191:                 collateralBalance - defaultedFee
192:             );
193:         }
194:
195:         _safeApprove(poolToken, address(this), poolBalance - fee);
196:         _safeTransferFrom(
197:             poolToken,
198:             address(this),
199:             msg.sender,
200:             poolBalance - fee
201:         );
```

Description

*****: In the withdraw function of the Pool contract, when the contract performs a transfer, it always calls the `_safeApprove` function for token approval before using `_safeTransferFrom` for token transfer. For different transfer targets of the same token, it involves multiple calls to the `_safeApprove` and `_safeTransferFrom` functions to approve and transfer the same token multiple times. However, it's important to note that the approval target is always `address(this)`, which means that calling the `_safeApprove` function multiple times to approve the same token to the current contract will consume more gas, which is unnecessary.

Recommendation

*****: It is recommended to perform only one complete `_safeApprove` operation when transferring the same token to save gas.
Consider the following fixes:

```
function withdraw() external nonReentrant onlyOwner afterExpiry {
    // code
    uint256 fee = 0;
+   _safeApprove(poolToken, address(this), poolBalance);

    /// @dev This is if the lender has made a profit
    if (poolBalance > poolDeposit) {
        // code

-       _safeApprove(poolToken, address(this), fee);
       // code

+       _safeApprove(collateralToken, address(this), collateralBalance);
        // code

-       _safeApprove(
-           collateralToken,
-           address(this),
-           collateralBalance - defaultedFee
-       );
        // code

-   _safeApprove(poolToken, address(this), poolBalance - fee);
    // code
}
```

Client Response

client response : Fixed. Fixed: lends-so/lends-p2p-lending@a435552

LDP-12:Redundant code

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	***

Code Reference

- code/contracts/Pool.sol#L91-L95
- code/contracts/Pool.sol#L118-L123

```
91: if (
92:             loans[msg.sender].collateralAmount > 0 ||
93:             loans[msg.sender].loanAmount > 0 ||
94:             loans[msg.sender].entryTimestamp > 0
95:         ) revert ActiveLoan();
```

```
118: loans[msg.sender] = Loan({
119:             borrower: msg.sender,
120:             entryTimestamp: block.timestamp,
121:             collateralAmount: loans[msg.sender].collateralAmount + _amount,
122:             loanAmount: loans[msg.sender].loanAmount + loanAmount
123:         });
```

Description

***: When user start to take loans, there is a validation to check if the user already has loans. So there is no need to accumulate user's `collateralAmount/loanAmount`.

Recommendation

***: Removing the redundant code.

```
        loans[msg.sender] = Loan({
            borrower: msg.sender,
            entryTimestamp: block.timestamp,
            collateralAmount: _amount,
            loanAmount: loanAmount
        });
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@96df2c6

LDP-13:Packed variables for `Pool` contract

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	***

Code Reference

- code/contracts/Pool.sol#L20C4-L29C38

```
uint256 public immutable apr;
    uint256 public immutable ltv;
    address public immutable collateralToken;
    address public immutable poolToken;
    address[] public whitelist;
    uint256 public expiry;

    uint256 public poolDeposit;
    uint256 public poolBorrowed;
    uint256 public collateralDeposit;
```

Description

*****:** In the pool contract uint256 is used for all variables which doesn't make sense for all of them because the storage used is huge in the contract, while they can be packed together.
Storing a uint256 costs 20,000 gas, while updating the value costs 5,000 gas. Smaller data types packed together reduce this cost because fewer 256-bit slots are used.
POC :
You can check on Foundry the gas used for deployment with these variables packed and not backed using forge test --gas-report

Recommendation

*****:** By packing variables into a single 256-bit slot, you reduce the total number of storage slots being used, which directly reduces gas costs during storage reads and writes.

- apr and ltv (Both uint256)

These values represent percentage-based amounts (APR and Loan-to-Value ratio). Given that they are likely expressed in basis points, they don’t need 256 bits.
Solution: Use `uint32` for each, as they can easily represent numbers up to ~4.29 billion.

- expiry:

If `expiry` is a Unix timestamp (usually measured in seconds), it can fit in `uint64` as the current timestamp is far from overflowing 64 bits.
Solution: Change `expiry` to `uint64` instead of `uint256`.
You can pack apr, ltv, and expiry into a single storage slot if they are changed to smaller sizes (uint32, uint32, uint64 respectively).
You can also pack poolDeposit, poolBorrowed, and collateralDeposit into two storage slots if they are reduced to uint128.
Optimized code :

```
uint256 public constant MAX_BASIS_POINTS = 10000;

IOracle public oracle;
IPoolFactory public poolFactory;

uint32 public immutable apr;                    // Reduced to 32 bits
uint32 public immutable ltv;                    // Reduced to 32 bits
uint64 public expiry;                           // Reduced to 64 bits
address public immutable collateralToken;
address public immutable poolToken;
address[] public whitelist;

uint128 public poolDeposit;                     // Reduced to 128 bits
uint128 public poolBorrowed;                    // Reduced to 128 bits
uint128 public collateralDeposit;               // Reduced to 128 bits
```

OPTIMIZATION ANALYSIS :
Total number of slots before optimization (excluding dynamic whitelist): 12 slots.
Total number of slots after optimization (excluding dynamic whitelist): 8 slots.

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@0be3392

LDP-14:Missing validation for `allBorrowers`

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/Pool.sol#L125

```
125: allBorrowers.push(msg.sender);
```

Description

***: There is no validation to check if `msg.sender` has already been added to `allBorrowers`, as the user can take loan multiple times.

Recommendation

***: Add validation to prevent `msg.sender` from being added multiple times to `allBorrowers`.

```
        uint256 borrowersLength = allBorrowers.length;
        bool isExists = false;
        for (uint256 i = 0; i < borrowersLength;) {
            if (allBorrowers[i] == msg.sender) {
                isExists = true;
                break;
            }
            unchecked{
                i++;
            }
        }
        if(!isExists){
            allBorrowers.push(msg.sender);
        }
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@cf2ee22

LDP-15:Insufficient Input Validation in Oracle Contract Constructor

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/Oracle.sol#L15-L19

```
constructor(TokenFeedPair[] memory _usdPairs) Ownable(msg.sender) {
        for (uint256 i = 0; i < _usdPairs.length; ++i) {
            usdTokenFeeds[_usdPairs[i].token] = _usdPairs[i].feed;
        }
    }
```

Description

*****:** The Oracle contract's constructor lacks proper input validation for the `_usdPairs` parameter, potentially allowing the initialization of the contract with invalid data. This oversight contrasts with the `addFeed` function, which includes checks for zero addresses and existing feeds.
Current implementation:

```
constructor(TokenFeedPair[] memory _usdPairs) Ownable(msg.sender) {
    for (uint256 i = 0; i < _usdPairs.length; ++i) {
        usdTokenFeeds[_usdPairs[i].token] = _usdPairs[i].feed;
    }
}
```

This implementation allows: 1. Setting of zero addresses for tokens or feeds. 2. Overwriting of existing feed addresses without checks.

Recommendation

*****:** Implement input validation in the constructor similar to the `addFeed` function:

```
constructor(TokenFeedPair[] memory _usdPairs) Ownable(msg.sender) {
    for (uint256 i = 0; i < _usdPairs.length; ++i) {
        if (_usdPairs[i].token == address(0) || _usdPairs[i].feed == address(0)) revert ZeroAddress();
        if (usdTokenFeeds[_usdPairs[i].token] != address(0)) revert FeedAlreadySet();
        usdTokenFeeds[_usdPairs[i].token] = _usdPairs[i].feed;
    }
}
```

Client Response

client response : Fixed. Completed: lends-so/lends-p2p-lending@ac521d1

Disclaimer