Blaize

Know the rules.
Create new ones

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

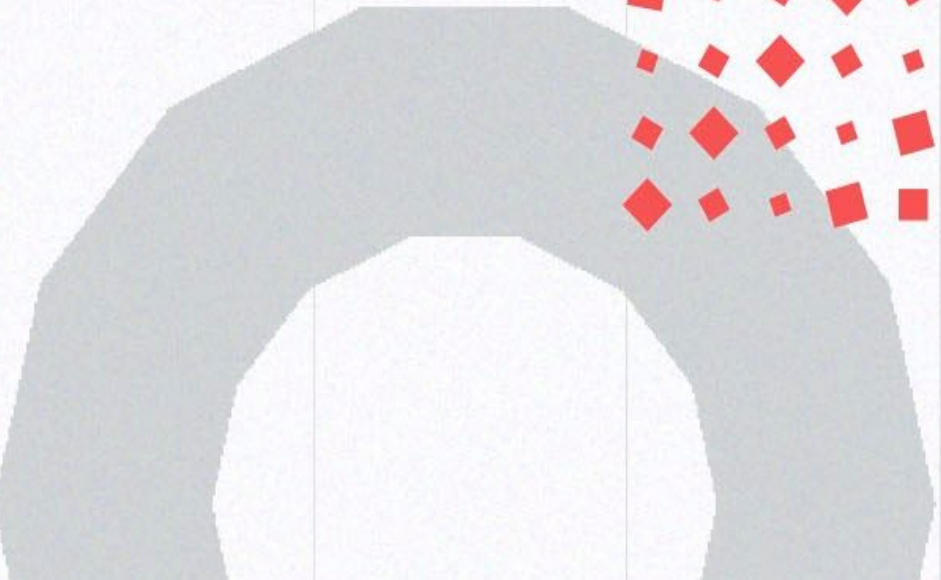


Table of Contents

This document may contain confidential information about IT systems and the intellectual property of the Customer and information about potential vulnerabilities and methods of their exploitation. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed - upon a decision of the Customer.

Abstract

In this report, we consider the security of the **Jibrel** contracts. Our task is to find and describe security issues in the smart contracts of the platform. This report presents the findings of the security audit of Customer's smart contracts conducted between **December 30th, 2020 - January 15th, 2021.**

Post-audit validation provided on **January, 21th, 2021**.

Disclaimer

The audit does not give any warranties on the security of the code. One audit can not be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audits are not investment advice.

Scope

The scope of the project is the whole **"tranche-protocol"** project at commit aa9cf5b23e4b8fcb0b01d605219ff2ad09871aae.

Post-audit scope for validation includes a **"tranche-protocol"** project at commit eb8f55c0bfeeaf3dee985ecf23cd40eb1e7a7c5a.

1. IJLoanHelper.sol
2. IJPriceOracle.sol
3. JFeesCollector.sol
4. JFeesCollector2.sol
5. JLoan.sol
6. JLoanHelper.sol
7. JLoanStorage.sol
8. JLoanStructs.sol
9. JPriceOracle.sol
10. Migrations.sol

11. myERC20.sol
12. TransferHelper.sol (TransferETHHelper.sol)
13. ITWAPOracle.sol
14. IJFeesCollector.so
15. IJLoan.sol
16. JFeesCollectorStorage.sol
17. JPriceOracleStorage.sol
18. OrFeedInterface.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited to):

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/local variables;
- Compile version not fixed.

Procedure

In our report we checked the contract with the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets best practices in efficient use of gas, code readability;

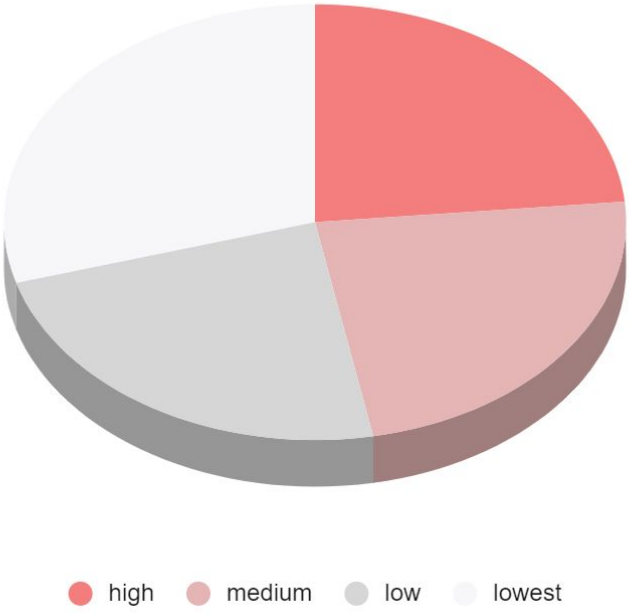
We perform our audit according to the following procedure:

1. Automated analysis:
    - Scanning contract by several public available automated analysis tools such as Mythril, Solhint, Slither and Smartdec;
    - Manual verification of all the issues found by tools.
2. Manual audit:
    - Manual analysis of smart contracts for security vulnerabilities;
    - Checking smart contract logic and comparing it with one described in the documentation.

Executive summary

According to the assessment, the Customer's smart contracts required improvements; some functionality works semi-auto and do not follow best practices. We described issues and added Customer comments according to the conclusion of these documents. Though, the Customer's team has provided all necessary improvements according to the Auditor's recommendations.

The graph of vulnerabilities distribution:



Severity Definition

Critical	A system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Needs immediate improvements and further checking.
High	A system contains a couple of serious issues, which lead to unreliable work of the system and might cause a huge information or financial leak. Needs immediate improvements and further checking.
Medium	A system contains issues which may lead to medium financial loss or users' private information leak. Needs immediate improvements and further checking.
Low	A system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Needs improvements.
Lowest	A system does not contain any issue critical to the secure work of the system, yet is relevant for best software defensive practices implementations.

AS-IS overview

JLoan contract overview

JLoan contract inherits from the JLoanStorage and OpenZeppelin's OwnableUpgrade contracts. Initialize function sets price oracle, fees collector and loan helper contracts addresses.

- From Initializable
  - isConstructor() (private)
- From ContextUpgradeSafe
  - __Context_init() (internal)
  - __Context_init_unchained() (internal)
  - _msgData() (internal)
  - _msgSender() (internal)
- From OwnableUpgradeSafe
  - __Ownable_init() (internal)
  - __Ownable_init_unchained() (internal)

    - owner() (public)
    - renounceOwnership() (public)
    - transferOwnership(address) (public)
- Native functions
    - updateVersion(uint256 _ver) (external) – updates contractVersion variable
    - setEarlySettlementWindow(uint256 _value) (external) – setter for generalLoansParams
    - setForeclosureWindow(uint256 _value) (external) - setter for generalLoansParams
    - setForeclosureRatio(uint8 _value) (external) - setter for generalLoansParams
    - setInstantForeclosureRatio(uint8 _value) (external) - setter for generalLoansParams
    - setRequiredCollateralRatio(uint8 _value) (external) - setter for generalLoansParams
    - setFactoryFees(uint8 _value) (external) – setter for generalLoanFees
    - setEarlySettlementFee(uint256 _value) (external) – setter for generalLoanFees
    - setUserRewardShare(uint8 _value) (external) – setter for generalLoanFees
    - setVaultShares(uint8 _value) (external) – setter for generalLoanFees
    - setUndercollateralizedForeclosingMultiple(uint16 _value) (external) – setter for generalLoanFees
    - setAtRiskForeclosedMultiple(uint16 _value) (external) – setter for generalLoanFees
    - setCancellationFees(uint8 _value) (external) – setter for generalLoanFees
    - getCollateralTokenAddress(uint256 _pairId) → address (public) – read from price oracle
    - getLentTokenAddress(uint256 _pairId) → address (public) – read from price oracle
    - getMinCollateralNoFeesAmount(uint256 _pairId, uint256 _askAmount) → uint256 (public) – calculation using JLoanHelper
    - getMinCollateralWithFeesAmount(uint256 _pairId, uint256 _askAmount) → uint256 (public) – calculation using JLoanHelper

- getMaxStableCoinNoFeesAmount(uint256 _pairId, uint256 _collAmount) → uint256 (public) – calculation using JLoanHelper
- getMaxStableCoinWithFeesAmount(uint256 _pairId, uint256 _collAmount) → uint256 (public) – calculation using JLoanHelper
- getCollFeesOnActivation(uint256 _collAmount) → uint256 (public) – calculation using JLoanHelper
- openNewLoan(uint256 _pairId, uint256 _borrowedAskAmount, uint256 _rpbRate) (external) – creates new loan, can receive ETH and ERC20 token for the loan, increases loanId by 1.
- getLoansCounter() → uint256 (external) – return loanId
- getGeneralParams() → struct JLoanStructs.GeneralParams (external) - return generalLoansParams
- getGeneralFees() → struct JLoanStructs.FeesParams (external) -return generalLoanFees
- depositEthCollateral(uint256 _id) (external) – send ETH to the contract, increase ETH collateral in the loan
- depositTokenCollateral(uint256 _id, address _tok, uint256 _amount) (external) - send token to the contract, increase token collateral in the loan
- withdrawCollateral(uint256 _id, uint256 _amount) (external) – withdraw collateral to the borrower
- getContractBalance(uint256 _id) → uint256 (external) – return contract balance on a token or eth
- getLoanBalance(uint256 _id) → uint256 (public) - return loanBalance[_id]
- getLoanStatus(uint256 _id) → uint256 (external) - return uint256(loanStatus[_id])
- setNewStatus(uint256 _id, uint256 _newStatus) (external) - set status for loan
- checkLoanInEarlySettlementWindow(uint256 _id) → bool (external) – check if loan is in early settlement period
- checkEarlySettledLoan(uint256 _id) → bool (external) - check if loan is in early settlement period
- setInitalCollateralRatio(uint256 _id) (external) - set initial collateral ratio of the loan
- getActualCollateralRatio(uint256 _id) → uint256 newCollRatio (public) - get the collateral ratio of the loan (subtracting the accrued interests)

- calcRatioAdjustingCollateral(uint256 _id, uint256 _amount, bool _adding) → uint256 ratio (external) - calculation using JLoanHelper
- calcDiffCollAmountForRatio(uint256 _id, uint256 _ratio) → uint256 collDiff (public) - calculation using JLoanHelper
- lenderSendStableCoins(uint256 _id, address _stableAddr) (external) - lender sends required stable coins to borrower
- setLoanStatusOnCollRatio(uint256 _id) → uint256 (public) - set the status of the loan based on collateral ratio
- initiateLoanForeclose(uint256 _id) (external) - set the loan in foreclosure state for undercollateralized loans
- setLoanForeclosing(uint256 _id) (internal) - set the loan in foreclosure state and set loanForeclosingBlock to the current block
- setLoanToForeclosed(uint256 _id) → bool (external) - set the loan in foreclosed state when foreclosureWindow time passed or collateral ratio is at risk
- setLoanForeclosed(uint256 _id) (internal) - set the loan in foreclosure state
- loanEarlyClosing(uint256 _id) → uint256 (internal) - set the loan in early closing state
- loanClosingByBorrower(uint256 _id) (external) - settle the loan in normal closing state by borrower
- borrowerSendBackLentToken(uint256 _id) (internal) - internal function for borrower to send back lent tokens to shareholders
- setLoanClosed(uint256 _id) (internal) - set the loan in closed state
- setLoanCancelled(uint256 _id) (external) - set the loan in cancelled state (only if pending)
- calculatingAccruedInterests(uint256 _id, uint256 _calcBlk) → uint256 (public) - calculate accrued interests of the contract
- getAccruedInterests(uint256 _id) → uint256 accruedInterests (public) - get accrued interests of the contract
- withdrawInterests(uint256 _id) → uint256 (public) - withdraw accrued interests for all shareholders and set the status after interests withdrawal
- withdrawInterestsMassive(uint256[] _id) → bool success (external) - withdraw accrued interests for a bunch of loans for all shareholders and set the status after interests withdrawal

- shareholderWithdrawInterests(uint256 _id, address _shareholder, uint256 _accruedTotalInterests) (internal) - withdraw accrued interests for a shareholder
- isShareholder(uint256 _id, address _holder) → bool (public) - check if an address is a shareholder
- getShareholderPlace(uint256 _id, address _holder) → uint256 (public) - get a shareholder place in shareholders array
- addLoanShareholders(uint256 _id, address _newShareholder, uint256 _amount) → uint256 (public) - add shareholder in shareholders arrays
- addLoanShareholdersMassive(uint256 _id, address[] _newShareholder, uint256[] _amount) → bool success (external) - add array of the shareholders in shareholders arrays
- addShareholderToMultipleLoans(uint256[] _ids, address _newShareholder, uint256[] _amounts) → bool success (external) - add one shareholder to multiple loans
- getSHAddress(uint256,uint256) (public) - get shareholder mapping based on shareholder number
- safeTransferCollateralAmounts(uint256,uint256,uint256) (internal) - transfers collateral
- setContractsAddress(address,address,address) (external) - set other contracts address

JLoanHelper contract overview

Inherits from the Ownable and IJLoanHelper.

- From Initializable
  - - isConstructor() (private)
- From ContextUpgradeSafe
  - - __Context_init() (internal)
  - - __Context_init_unchained() (internal)
  - - _msgData() (internal)
  - - _msgSender() (internal)
- From OwnableUpgradeSafe
  - __Ownable_init() (internal)
  - __Ownable_init_unchained() (internal)
  - owner() (public)
  - renounceOwnership() (public)
  - transferOwnership(address) (public)
- Native functions

- adjustDecimalsCollateral(uint256,uint256,uint256) (public) - adjust for decimals in tokens pair for collateral
- adjustDecimalsRatio(uint256,uint256,uint256) (internal) - adjust for decimals in tokens pair for ratio
- calcMaxStableCoinAmount(uint256,uint256,uint8) (public) - get the amount of stable coin that a borrower could receive in front of a collateral amount
- calcMaxStableCoinWithFeesAmount(uint256,uint256,uint8,uint8) (external) - get the amount of stable coin that a borrower could receive in front of a collateral amount with activation fees
- calcMinCollateralAmount(uint256,uint256,uint8) (public) - get the amount of collateral needed to have stable coin amount
- calcMinCollateralWithFeesAmount(uint256,uint256,uint8,uint8) (public) - get the amount of collateral needed to have stable coin amount, with fees
- calculateCollFeesOnActivation(uint256,uint8) (public) - calculate fees on collateral amount
- collateralAdjustingRatio(uint256,uint256,uint256,uint256,bool) (external) - calc a new ratio if collateral amount has added to contract balance
- constructor(address) (public) -
- getCollateralRatio(uint256,uint256,uint256) (external) - get the collateral ratio of the loan (subtracting the accrued interests)
- ratioDiffCollAmount(uint256,uint256,uint256,uint256) (external) - calc how much collateral amount has to be added to have a ratio
- roundDn(uint256,uint256,uint256) (internal) - divides and mathematically incorrect rounds down
- roundUp(uint256,uint256,uint256) (internal) – divides and mathematically incorrect rounds up
- calcAccruedInterests(uint256,uint256,uint256) (internal) - calculate accrued interests of the contract
- calcActualCollateralRatio(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256) (public) - get the collateral ratio of the loan
- calcLoanStatusOnCollRatio(uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint8,uint8) (external) - set the status of the loan based on collateral ratio
- getAccruedInterests(uint256,uint256,uint256,uint256,uint256,uint256,uint256) (public) - get accrued interests of the loan

JPriceOracle contract overview

Inherits from the OwnableUpgradeSafe and IJPriceOracle.

- From Initializable
  - isConstructor() (private)
- From ContextUpgradeSafe
  - __Context_init() (internal)
  - __Context_init_unchained() (internal)
  - _msgData() (internal)
  - _msgSender() (internal)
- From OwnableUpgradeSafe
  - __Ownable_init() (internal)
  - __Ownable_init_unchained() (internal)
  - owner() (public)
  - renounceOwnership() (public)
  - transferOwnership(address) (public)
- Native functions
  - _addAdmin(address) (internal) – add admin address
  - _removeAdmin(address) (internal) – remove admin address
  - addAdmin(address) (external) – add admin address
  - getPairBaseAddress(uint256) (external) - get a pair base address
  - getPairBaseDecimals(uint256) (external) - get a pair base decimals
  - getPairCounter() (external) - get a pair counter
  - getPairDecimals(uint256) (external) - get a pair decimals
  - getPairName(uint256) (external) - get a pair name
  - getPairQuoteAddress(uint256) (external) - get a pair quote address
  - getPairQuoteDecimals(uint256) (external) - get a pair quote decimals
  - getPairValue(uint256) (external) - get a pair price
  - initialize() (public)
  - isAdmin(address) (public) – check if address is admin
  - removeAdmin(address) (external) – remove admin
  - renounceAdmin() (external) – renounce admin role
  - setBaseQuoteDecimals(uint256,uint8,uint8) (external) - set a base and quote decimals for the specified pair
  - setNewPair(string,uint256,uint8,address,uint8,address,uint8) (external) - set a new pair

- setPairValue(uint256,uint256,uint8) (external) - set a price for the specified pair
- updateVersion(uint256) (external) - update contract version
- bytes32ToString(bytes32) (public) - helper, casts bytes32 to the string
- fixed_1() (public) - returns 10^24
- getChainlinkDecimals(uint256) (public) - get latest decimals of a single pair from chainlink
- getChainlinkDescription(uint256) (public) - get latest description of a single pair from chainlink
- getChainlinkPrice(uint256) (public) - get latest price of a single pair from chainlink
- getLatestChainlinkPairInfo(uint256) (external) - get latest info on single pair from chainlink
- getOrFeedPrice(uint256,string,uint256) (public) - get price from orFeed
- getUniswapPrice(uint256,uint256) (public) - get price from uniswap
- getUniswapTimeWeightedAveragePrice(uint256,uint256) (public) -get time weighted average price from uniswap
- setExternalProviderParameters(uint256,address,uint8,bool) (external) - set a chainlink parameters for the specified pair
- setOrFeedAddress(address) (external) - set orFeed address
- setUniswapRouterAddress(address) (external) - set uniswap router address

JFeesCollector contract overview

Inherits from the OwnableUpgradeSafe

- From Initializable
  - isConstructor() (private)
- From ContextUpgradeSafe
  - __Context_init() (internal)
  - __Context_init_unchained() (internal)
  - _msgData() (internal)
  - _msgSender() (internal)

- From OwnableUpgradeSafe
  - __Ownable_init() (internal)
  - __Ownable_init_unchained() (internal)
  - owner() (public)
  - renounceOwnership() (public)
  - transferOwnership(address) (public)
- Native functions
  - allowToken(address) (external) - add allowed token address
  - disallowToken(address) (external) - remove allowed token address
  - ethWithdraw(uint256) (external) - withdraw eth amount
  - getEthBalance() (external) - get eth contract balance
  - getTokenBalance(address) (external) - get contract token balance
  - initialize() (public)
  - isTokenAllowed(address) (public) - check if a token is already allowed
  - receive() (external) – fallback, emits event EthReceived
  - updateVersion(uint256) (external) - update contract version
  - withdrawTokens(address,uint256) (external) - withdraw tokens from the contract, checking if a token is already allowed

TransferHelper (TransferETHHelper) contract overview

- From TransferHelper (TransferETHHelper)
  - safeApprove(address,address,uint256) (internal) – safe approve helper
  - safeTransfer(address,address,uint256) (internal) – safe transfer helper
  - safeTransferETH(address,uint256) (internal) – safe transfer eth helper
  - safeTransferFrom(address,address,address,uint256) (internal) – safe transfer token from address helper

Audit overview

Critical

No critical issues detected.

High

1. **JLoanHelper.sol** `calcMaxStableCoinAmount` **(#117)**: subtraction overflows if base decimals is less than quote decimals.

   Evidence: Alternative branch (#116) with `baseDecimals.sub(quoteDecimals)` always fails, because `baseDecimals` is strictly less than `quoteDecimals` in this branch.

```
function calcMaxStableCoinAmount(uint256 _pairId, uint256 _collAmount, uint8 _requiredCollateralRatio) public override view returns (uint256) {
    uint256 price = IJPriceOracle(priceOracleAddress).getPairValue(_pairId);
    uint256 pairDecimals = uint256(IJPriceOracle(priceOracleAddress).getPairDecimals(_pairId));
    uint256 askAmount = roundDn(_collAmount.mul(100).mul(price).div(uint256(_requiredCollateralRatio)), 10 ** pairDecimals, 0);
    uint256 baseDecimals = uint256(IJPriceOracle(priceOracleAddress).getPairBaseDecimals(_pairId));
    uint256 quoteDecimals = uint256(IJPriceOracle(priceOracleAddress).getPairQuoteDecimals(_pairId));
    if (baseDecimals ≥ quoteDecimals) {
        uint256 diffBaseQuoteDecimals = baseDecimals.sub(quoteDecimals);
        askAmount = askAmount.div(10 ** diffBaseQuoteDecimals).sub(5); //subtract 5 to be sure everything is ok
    } else {
        uint256 diffBaseQuoteDecimals = baseDecimals.sub(quoteDecimals);
        askAmount = askAmount.mul(10 ** diffBaseQuoteDecimals).sub(5); //subtract 5 to be sure everything is ok
    }
    return askAmount;
}
```

   Recommendation: replace baseDecimals.sub(quoteDecimals) with quoteDecimals.sub(baseDecimals) in JLoanHelper.sol at line 117.

   Resolution: Fixed by the Customer's team.

2. **JLoanHelper.sol** `roundUp` **(#35)** and `roundDn` **(#48):** round down and round up are mathematically incorrect and `roundDn` always revert on values such as (1, 10, 0).

   Evidence:

```
35    function roundUp(uint256 numerator, uint256 denominator, uint256 precision) internal pure returns (uint256) {
36        uint256 _numerator  = numerator.mul(10 ** (precision.add(1)));
37        uint256 _quotient =  ((_numerator.div(denominator)).add(5)).div(10);
38        return _quotient;
39    }
```

```
48    function roundDn(uint256 numerator, uint256 denominator, uint256 precision) internal pure returns (uint256) {
49        uint256 _numerator  = numerator.mul(10 ** (precision.add(1)));
50        uint256 _quotient =  (_numerator.div(denominator).sub(5)).div(10);
51        return _quotient;
52    }
```

   Recommendation: rename functions to ceil() and floor() and rewrite them as follows:

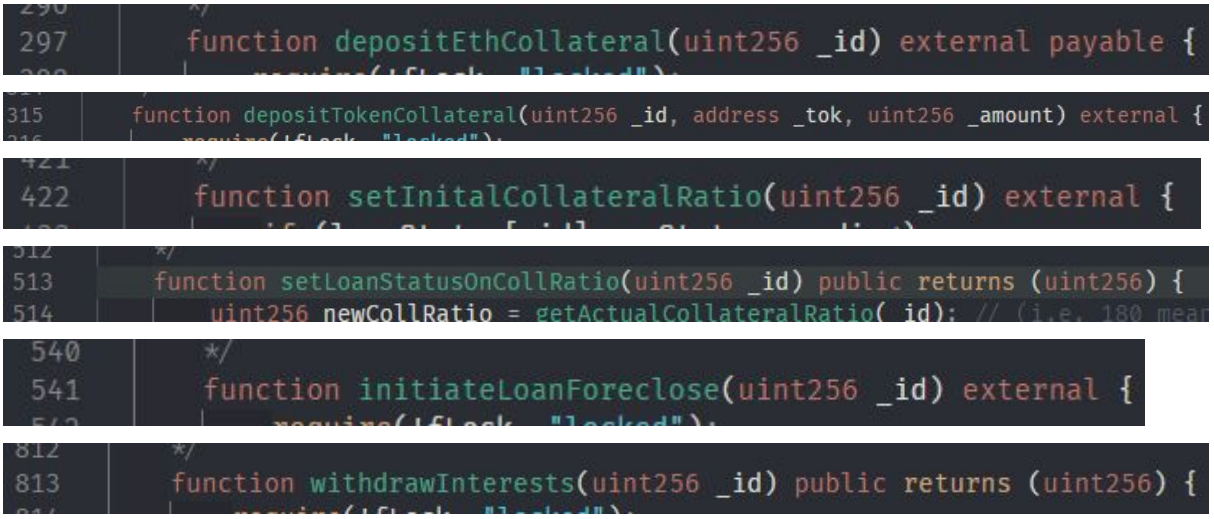
```
function roundUp(uint256 numerator, uint256 denominator, uint256 precision) internal pure returns (uint256) {
    uint256 _numerator  = numerator.mul(10 ** precision);
    return denominator.sub(1).add(_numerator).div(denominator);
}

function roundDn(uint256 numerator, uint256 denominator, uint256 precision) internal pure returns (uint256) {
    uint256 _numerator  = numerator.mul(10 ** precision);
    return _numerator.div(denominator);
}
```

Resolution: Fixed by the Customer's team.

3. **JLoan** functions (#813,#513,#541,#422,#315,#297). Anybody can call functions that change storage.

   Evidence:



   Recommendation: check that it's safe to allow arbitrary addresses to call these functions and add necessary `require()` statements.

   Resolution: Customer's team has confirmed the correctness of the logic.

4. **TransferHelper.sol** `safeApprove()` (#6) safeApprove() method in the TransferHelper is vulnerable to the front running.

   Evidence: It creates potential for an approved user to spend more than the intended amount if call transferFrom() both before and after the call to approve().

```
6     function safeApprove(address token, address to, uint256 value) internal {
7         // bytes4(keccak256(bytes('approve(address,uint256)')));
8         (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
9         require(success && (data.length == 0 || abi.decode(data, (bool))), 'TH APPROVE_FAILED');
10    }
```

Recommendation: OpenZeppelin's safeERC20 wrapper should be added instead of the **TransferHelper.sol** or add `increaseApprove()` and `decreaseApprove()` functionality.

Resolution: Fixed by the Customer's team.

Medium

1. **JLoan.sol** `setLoanStatusOnCollRatio()` (#513) Event `LoanStatusChanged` is emitted even if there is no status change.

Evidence:

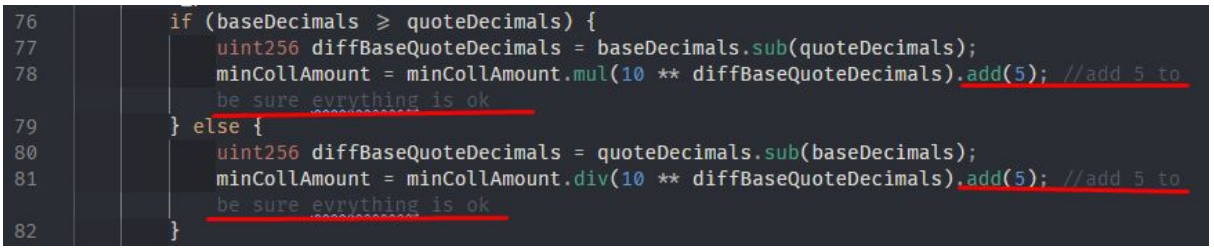
```
513     function setLoanStatusOnCollRatio(uint256 _id) public returns (uint256) {
514         uint256 newCollRatio = getActualCollateralRatio(_id); // (i.e. 180 means 180%)
515         uint256 oldStatus = uint256(loanStatus[_id]);
516         if (oldStatus > 0 && oldStatus < 4) {      //not allowed if loan not active or fore
517             if ( newCollRatio ≥ generalLoansParams.foreclosingRatio ) {
518                 loanStatus[_id] = Status(1);
519                 loanForeclosingBlock[_id] = 0;    //reset foreclosing block
520             } else if ( newCollRatio ≤ generalLoansParams.foreclosingRatio && newCollRatio
521                 loanStatus[_id] = Status(2);
522             } else if ( newCollRatio < generalLoansParams.instantForeclosureRatio ) {
523                 loanStatus[_id] = Status(3);
524             }
525         } else if (oldStatus == 4) {
526             if ( newCollRatio ≥ generalLoansParams.foreclosingRatio ) {
527                 loanStatus[_id] = Status(1);
528                 loanForeclosingBlock[_id] = 0;    //reset foreclosing block
529             }
530         }
531         // else no change on status
532         emit LoanStatusChanged(_id, oldStatus, uint256(loanStatus[_id]), 0, 0);
533         return uint256(loanStatus[_id]);
534     }
```

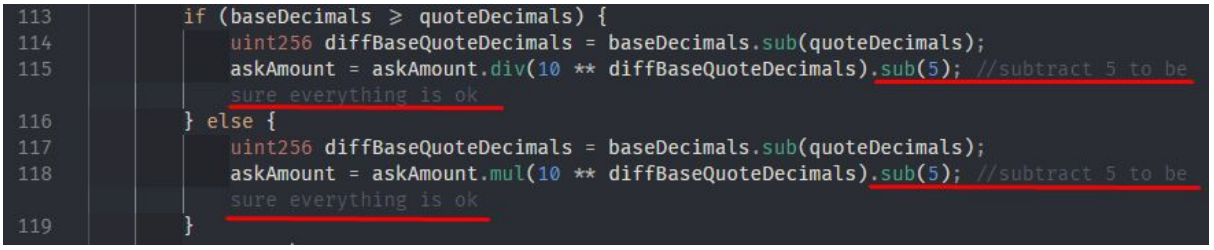
Recommendation: Restrict event emission.

Resolution: Fixed by the Customer's team.

2. **JLoanHelper.sol** `calcMinCollateralAmount` (#78 and #81) and `calcMaxStableCoinAmount` (#115 and #118). Vague documentation and unclear, potentially unnecessary math.

Evidence:



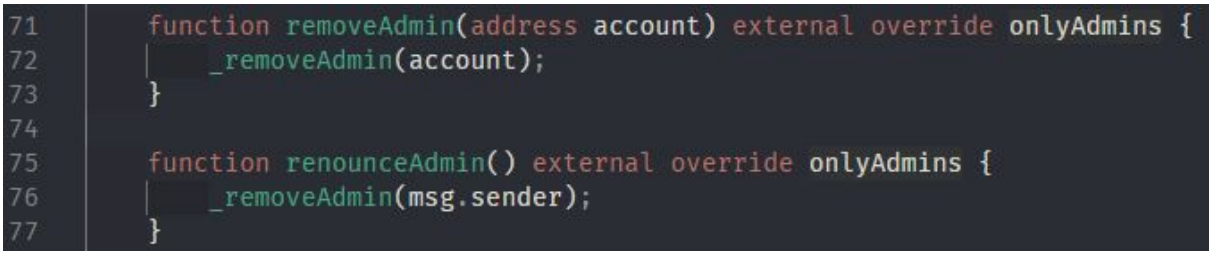


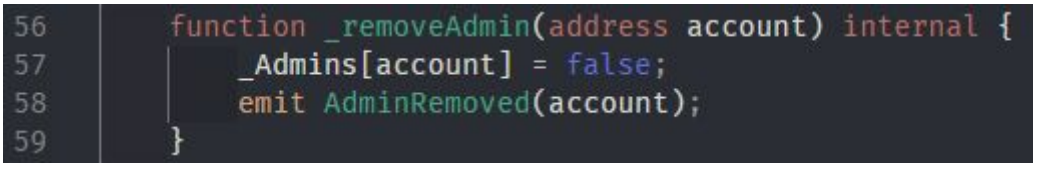
Recommendation: check calculations and add documentation to these methods.

Resolution: Customer's team has confirmed the correctness of the logic.

3. **JPriceOracle.sol** `renounceAdmin (#75)` and `removeAdmin (#71)`. It's possible to use these functions to leave the contract without any admins.

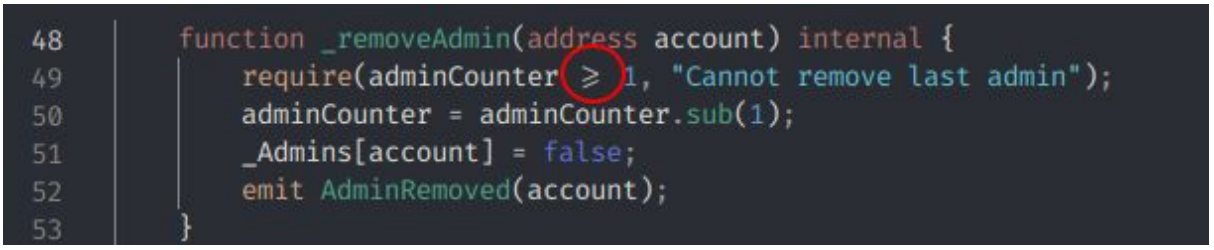
Evidence:

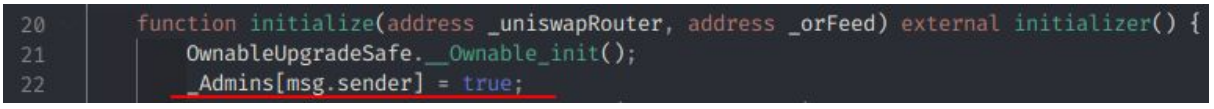




Recommendation: Check and decide if this functionality is acceptable for you. If not then add a default admin address or check that the last administrator will not be removed.

Addition: Wrong admin removing `require` - there must be strictly more than 1 admin. In the initialize function admin added using mapping, not `_addAdmin` function.





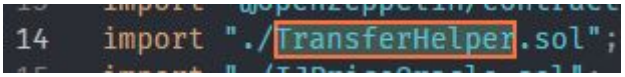
Recommendation: change `require` to the `require(adminCounter > 1, "Cannot remove last admin")`; and use `_addAdmin` function for adding new admin in the `initialize`.

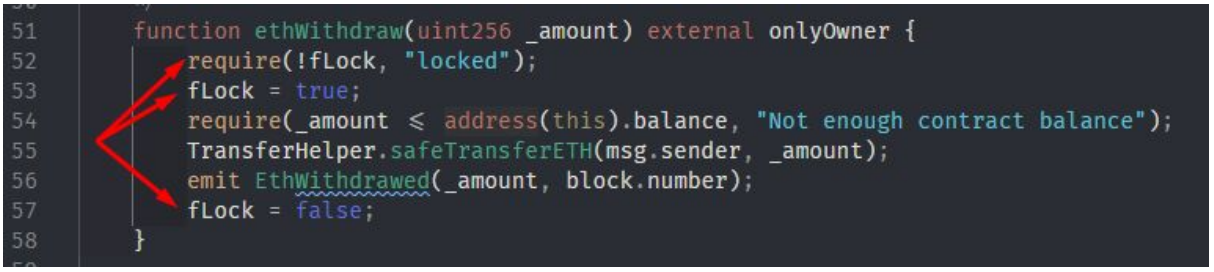
Resolution: Fixed by the Customer's team.

Low

1. Redundant custom contracts.

Evidence:





Addition: flock is still present in the following functions (**JFeesCollector**):

```
42    function ethWithdraw(uint256 _amount⬆) external onlyOwner {
43        require(!fLock, "locked");
44        fLock = true;
45        require(_amount⬆ ⩽ address(this).balance, "Not enough contract balance");
46        TransferETHHelper.safeTransferETH(msg.sender, _amount⬆);
47        emit EthWithdrawn(_amount⬆, block.number);
48        fLock = false;
49    }
```

```
102   function withdrawTokens(address _tok⬆, uint256 _amount⬆) external onlyOwner {
103       require(!fLock, "locked");
104       fLock = true;
105       require(isTokenAllowed(_tok⬆), "Token not allowed");
106       SafeERC20.safeTransfer(IERC20(_tok⬆), msg.sender, _amount⬆);
107       emit TokenWithdrawed(_tok⬆, _amount⬆, block.number);
108       fLock = false;
109   }
```

Recommendation: You can use OpenZeppelin's safe ERC20 wrapper instead of the TransferHelper.sol. You can use OpenZeppelin's ReentrancyGuard instead of flock for the non-reentrant functions.

Resolution: Fixed by the Customer's team.

2. **JPriceOracle** (#133, #191, #201) Documentation discrepancies.

Evidence:

a. "price" must be changed to the "counter"

```
132     /**
133     * @dev get a pair price      hg-tranche, 2 months ago • loan contracts
134     * @return pairs counter
135     */
136     function getPairCounter() external override view returns (uint256) {
137         return pairCounter;
138     }
139
```

b. "decimals" must be changed to the "address":

```
190     /**
191     * @dev get a pair base decimals
192     * @param _pairId number of the pair
193     * @return address of base currency decimals
194     */
195     function getPairBaseAddress(uint256 _pairId) external override view returns (address) {
196         require(_pairId < pairCounter, "pair does not exists");
197         return pairs[_pairId].baseAddress;
198     }
```

c. "decimals" must be changed to the "address":

Recommendation: fix documentation comments.

Resolution: Fixed by the Customer's team.

3. **JLoan.sol** `getLoansCounter` (#272), `getGeneralParams` (#281), `getGeneralFees` (#289), `getLoanBalance` (#375), `getLoanStatus` (#383) There are redundant external getters for loanId, generalLoansParams, generalLoanFees, loanBalance[_id] and loanStatus[_id] variables – they all are public, so getters are generated automatically.

   Evidence:

```
272     function getLoansCounter() external  view returns (uint256) {
273         return loanId;
274     }
```

```
20      // loan id
21      uint256 public loanId;
```

```
281     function getGeneralParams() external view returns (GeneralParams memory) {
282         return generalLoansParams;
283     }
```

```
40      // general fees parameters
41      FeesParams public generalLoanFees;
42
```

```
289     function getGeneralFees() external view returns (FeesParams memory) {
290         return generalLoanFees;
291     }
```

```
40      // general fees parameters
41      FeesParams public generalLoanFees;
```

```
374     function getLoanBalance(uint256 _id) public view returns (uint256) {
375         return loanBalance[_id];
376     }
```

```
50      mapping(uint256 => uint256) public loanBalance;
```

```
383     function getLoanStatus(uint256 _id) external view returns (uint256) {
384         return uint256(loanStatus[_id]);
385     }
```

```
45      mapping (uint256 => Status) public loanStatus;
```

Recommendation: remove getters.

Resolution: Fixed by the Customer's team.

4. **JLoan.sol** `getContractBalance` (#361), `checkLoanInEarlySettlementWindow` (#401) and `checkEarlySettledLoan` (#414) These getters are redundant, their values can be calculated off-chain.

Evidence:

```
356     /**
357     * @dev get contract overall balance about a token or eth
358     * @param _id loan id
359     * @return balance
360     */
361     function getContractBalance(uint256 _id) external view returns (uint256) {
362         address collateralToken = getCollateralTokenAddress(loanPair[_id]);
363         if (collateralToken == address(0))
364             return address(this).balance;
365         else
366             return IERC20(collateralToken).balanceOf(address(this));
367     }
```

```
396     /**
397     * @dev check if loan is in early settlement period
398     * @param _id loan id
399     * @return boolean
400     */
401     function checkLoanInEarlySettlementWindow(uint256 _id) external view returns (bool) {
402         uint256 lastEarlyBlock = loanActiveBlock[_id].add(generalLoansParams.earlySettlementWindow);
403         if (block.number <= lastEarlyBlock)
404             return true;
405         else
406             return false;
407     }
```

```
409     /**
410     * @dev check if loan is in early settlement period
411     * @param _id loan id
412     * @return boolean
413     */
414     function checkEarlySettledLoan(uint256 _id) external view returns (bool) {
415         return loanStatus[_id] == Status.earlyClosing;
416     }
```

Recommendation: Check the necessity of these getters and remove redundant ones.

Resolution: Customer's team has confirmed the correctness of the logic.

5. JPriceOracle.sol `fixed_1` (#194) This function can be replaced with the `public constant` variable for gas saving.

   Evidence:

```
194     function fixed_1() public pure returns(uint256) {
195         return 1000000000000000000000000;
196     }
```

   Recommendation: Replace this function with the `public constant` variable.

   Resolution: Fixed by the Customer's team.

6. JPriceOracle.sol `reciprocal` (#218) `assert` is gas ineffective.

   Evidence:

```
218     function reciprocal(uint256 x⬆) public pure returns (uint256) {
219         assert(x⬆ ≠ 0);
220         return (fixed_1()*fixed_1()).div(x⬆); // Can't overflow
221     }
```

   Recommendation: Change `assert` to the `require` statement.

   Resolution: Fixed by the Customer's team.

7. Obfuscate tests' system.

   Evidence:
   The system is designed in a way that tests are run on-by-one instead of running the complete testset by a `truffle test` command.

Recommendation: Truffle framework is the industry standard for the contracts development and testing. And mixed environment, non-standard interaction with the project may lead to the obfuscation, impossibility of coverage verification and complicated development.

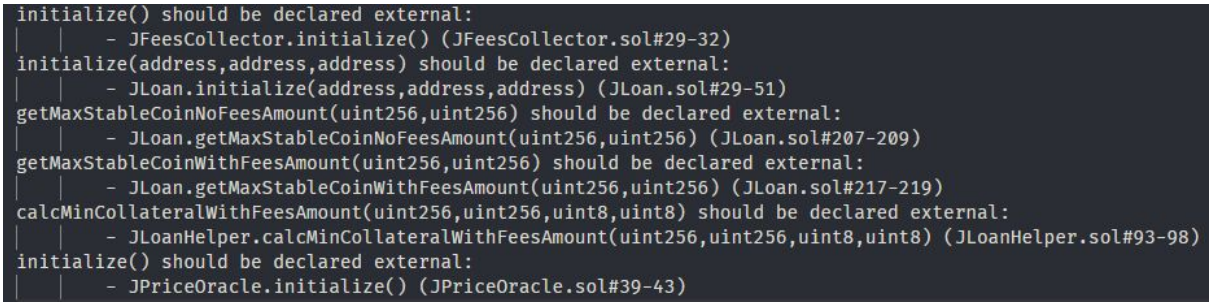
We recommend you to rewrite tests to run them from the `'truffle test'` command and change your test script in the package.json.

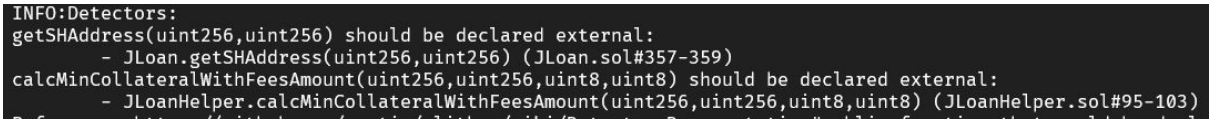
Lowest

Informational statements

1. Functions should be declared external.

   Evidence:





   Recommendation: declare these functions as external.

   Resolution: Fixed by the Customer's team.

2. **JPriceOracle.sol** (#39-77) Documentation is missing.

   Evidence:

```
39      function initialize() public initializer() {
40          OwnableUpgradeSafe.__Ownable_init();
41          _Admins[msg.sender] = true;
42          contractVersion = 1;
43      }
44
45      modifier onlyAdmins() {
46          require(isAdmin(msg.sender), "!Admin");
47          _;
48      }
49
50      /*   Admins Roles Mngmt   */
51      function _addAdmin(address account) internal {
52          _Admins[account] = true;
53          emit AdminAdded(account);
54      }
55
56      function _removeAdmin(address account) internal {
57          _Admins[account] = false;
58          emit AdminRemoved(account);
59      }
60
61      function isAdmin(address account) public override view returns (bool) {
62          return _Admins[account];
63      }
64
65      function addAdmin(address account) external override onlyAdmins {
66          require(account != address(0), "Not a valid address!");
67          require(!isAdmin(account), " Address already Administrator");
68          _addAdmin(account);
69      }
70
71      function removeAdmin(address account) external override onlyAdmins {
72          _removeAdmin(account);
73      }
74
75      function renounceAdmin() external override onlyAdmins {
76          _removeAdmin(msg.sender);
77      }
78
```

Recommendation: add documentation for each function.

Resolution: Fixed by the Customer's team.

3. **JLoan.sol** `initialize` (#29), `setForeclosureRatio` (#87), `onlyAdmins` (#53) and `fallback` (#58) Missing documentation.

Evidence:

```
27      event InterestsWithdrawed(uint256 id, uint256 accruedInterests);
28
29      function initialize(address _priceOracle, address _feesCollector, address _loanHelper) public initializer() {
30          OwnableUpgradeSafe.__Ownable_init();
```

```
86
87      function setForeclosureRatio(uint8 _value) external onlyAdmins {
88          generalLoansParams.foreclosingRatio = _value;
89      }
```

```
52
53      modifier onlyAdmins() {
54          require(IJPriceOracle(priceOracleAddress).isAdmin(msg.sender), "!Admin");
55          _;
56      }
57
```

```
57
58      fallback() external { // cannot deposit eth
59          revert("ETH not accepted!");
60      }
61
```

Recommendation: add documentation.

Resolution: Fixed by the Customer's team.

4. Redundant default files.

   Evidence: Migrations.sol

   Recommendation: You can remove Migrations.sol since it provides no efforts for the project.

   Resolution: Fixed by the Customer's team.

5. Obfuscated project structure.

   a. *Evidence*:
   Contracts used only for testing in the same directory with production smart contracts.

   Recommendation: Move myERC20 and JFeesCollector2 contracts to the `contracts/test` directory.

   b. *Evidence*:
   In the `contracts/uniswap` directory only the ITWAPOracle contract is used in contracts.

Recommendation: Move all the other contracts from the `contracts/uniswap` directory to the `contracts/mocks` directory and leave only contracts in use (ITWAPOracle.sol).

Resolution: Fixed by the Customer's team.

6. **JLoan.sol** (#413) Documentation is missing.

Evidence:

```
413     function safeTransferCollateralAmounts(uint256 _id, uint256 _userReward, uint256 _vaultReward) internal {
```

Resolution: Fixed by the Customer's team.

Unit Test Coverage

The project test framework is obfuscated with a non-standard and purely documented approach, where tests are running one-by-one instead of a complete suite. Though all present tests can be successfully run. Nevertheless, the non-standard approach does not allow to check the test coverage in an automatic way, so manual review for tests coverage was applied. The Auditor's team has considered that the project has sufficient test coverage.

Conclusion

According to the audit the contract was manually reviewed and analyzed with static analysis tools. The Audit team has found some high, medium and low issues during the analysis. Though, all issues were fixed by the Customer's team following Auditor's recommendations. Nevertheless, the custom non-standard approach for unit testing obfuscates the project development in general and the audit process in particular.

The overall security of the smart-contracts system can be evaluated as **Highly Secure**, **95** out of **100**.

Audit report contains all necessary information related to it as well as recommendations for their elimination.