Blaize

Know the rules.
Create new ones

# SMART CONTRACT TECHNICAL EXPERTISE

# 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 general quality of the **Jibrel** project. Our task is to perform manual code review and technical expertise over the project. This report presents the findings of the analysis of Customer's project conducted between **December 30th, 2020 - January 15th, 2021**.

Post-audit validation provided on **January, 19th, 2021**.

## Scope

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

Post-audit scope for validation includes **"tranche-protocol"** project at commit 44ffcb35830a7a3182d1fd944f6a887c600b2eb0.

We have scanned this project for common development practices. Here are some reviews we conducted (the full list includes them but is not limited to):

- General code review
- Developer tools usage review
- Test coverage review
- Storage variables usage analysis
- Dependency review
- Gas cost analysis

## Executive summary

According to the assessment, the Customer's smart contracts **required improvements; some functionality worked semi-auto and did not follow best practices.** We described issues and added recommendations

for their elimination. Also, the Customer's team **has provided** all necessary improvements according to the Auditor's recommendations.

## 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. |
| **Informational** | A system does not contain any issue critical to the secure work of the system, yet is relevant for best software defensive practices implementations. |

## Technical expertise overview

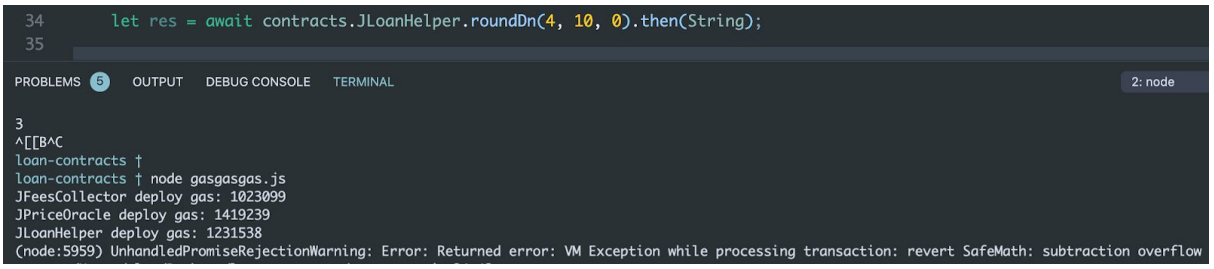
### Critical

No critical issues detected.

### High

1. Functions `roundUp()` and `roundDn()` in **JLoanHelper.sol** perform incorrect calculations (`JLoanHelper.sol`, lines 35 and 48)

   *Evidence*: These functions fail, for example, at value (4, 10, 0).




*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.

2. Incorrect calculations in `JLoanHelper::calcMaxStableCoinAmount` as a result of copy-paste error.

   *Evidence*: In `else` branch, `baseDecimals.sub(quoteDecimals)` always fails, because `baseDecimals` is strictly less than `quoteDecimals` in this branch:

```
if (baseDecimals >= quoteDecimals) {
    uint256 diffBaseQuoteDecimals = baseDecimals.sub(quoteDecimals);
    askAmount = askAmount.div(10 ** diffBaseQuoteDecimals).sub(5); //
} else {
    uint256 diffBaseQuoteDecimals = baseDecimals.sub(quoteDecimals);
    askAmount = askAmount.mul(10 ** diffBaseQuoteDecimals).sub(5); //
}
```

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

   *Resolution*: Fixed by the Customer's team.

### Medium

1. `fLock` reentrancy guard is error-prone.

   *Evidence*: `fLock = true` must always be at the beginning of the function, and `fLock = false` always in the end. In `JLoan::initiateLoanForeclose` there is a statement `fLock = false` in the middle of the function (`JLoan.sol`, line 557). There is a possibility for failure because of a function structure mistake in the case of a return statement in the middle of a function.

   *Recommendation*: use the Openzeppelin's `ReentrancyGuard` or a modifier like:

   ```
   modifier locked() {
       require(!fLock, "locked");
       fLock = true;
       _;
       fLock = false;
   }
   ```

   *Resolution*: Fixed by the Customer's team.

2. `JLoanStorage::loanInitiateForecloseBlock` storage variable is redundant (`JLoanStorage.sol`, line 56).

   *Evidence*: It's only set once and never used, though `loanForeclosingBlock` is present in several blocks of the code.

   *Recommendation*: remove or rename and reuse this variable.

   *Resolution*: Fixed by the Customer's team.

3. Possible numeric error in `JLoan::loanEarlyClosing` (`JLoan.sol`, line 667)

   *Evidence:* If `lastInterestWithdrawalBlock[_id] != 0`, `balanceRequested` becomes

`generalLoansParams.earlySettlementWindow - blockAlreadyUsed - blockAlreadyUsed`. Double subtraction of the same value.

*Recommendation*: check calculations and make computations more clear, for example:

```
if (lastInterestWithdrawalBlock[_id] != 0) {
    uint256 blockAlreadyUsed = lastInterestWithdrawalBlock[_id].sub(loanActiveBlock[_
    uint256 remainingBlock = (generalLoansParams.earlySettlementWindow).sub(blockAlre
    return remainingBlock.mul(loanParams[_id].rpbRate);
} else {
    uint256 remainingBlock = generalLoansParams.earlySettlementWindow;
    return remainingBlock.mul(loanParams[_id].rpbRate);
}
```

*Resolution*: Fixed by the Customer's team.

**Low**

1. Several tests are incorrect.

*Evidence*: Tests "`borrower1 can send collateral to set loan0 back in active state, but not enough Eth`" in both `JLoansForeclosedByRatio.test.js` (line 89) and `JLoansForeclosedByTime.test.js` (line 89) check if one can send more ether than he has on balance, which always fails.

```
// Of course if he doesn't have enough funds, transaction will revert:
// (code modified to fit in a screenshot)
await expectRevert(
  this.JLoan.depositEthCollateral(0, {from: borrower1, value: collToAdd}),
  "Returned error: sender doesn't have enough funds to send tx. The upfront cost
);
```

*Recommendation*: rewrite these tests in a way where a test sends less ether than is expected by the contract.

**Informational**

*Code style issues*

1. In JLoan.sol, line 403:
   The block:

```
if (block.number <= lastEarlyBlock)
    return true;
else
    return false;
```

   can be replaced with:

```
return block.number <= lastEarlyBlock;
```

*Informational statements*

1. `Migrations.sol` and `1_initial_migration.js` files are never used and should be removed.

   *Resolution*: Fixed by the Customer's team.

2. Typo in `JLoan.sol`, line 272: "*conuter*" instead of "*counter*".
   Typo in `JLoan.sol`, line 474: "*stabl*" instead of "*stable*".

   *Resolution*: Fixed by the Customer's team.

3. Several checks in tests are redundant. The common form is:

```
expect(tx.receipt.transactionHash).to.match(/0x[0-9a-fA-F]{64}/);
```

   They are always true and only obfuscate the code.

   *Resolution*: Fixed by the Customer's team.

4. Every developer's environment is different. This can make every action harder to reproduce between machines. Consider removing `package-lock.json` (and `yarn.lock`) from `gitignore`. With these files in git, every developer in the team will have the same dev environment, which can save a lot of time.

   *Resolution*: Fixed by the Customer's team.

5. Contract flattening is never used in the project. In spite of files `flatten.sh` and `flatten.bat`, there is no **`dist`** folder in `gitignore`. Though, the contents of the **`dist`** folder are never used.

   Recommendation: add `dist` folder to `gitignore`, and consider removing flattening altogether and replace it with `@resolver-engine/imports-fs` and `@resolver-engine/imports` libraries, which can be used to publish source code on Etherscan more reliably.

6. Redundant logs in tests. The statement `gasUsed * gasPrice` is logged in several tests. These logs only contribute to code obfuscation. Gas price changes constantly, so these logged tx costs won't be helpful. Consider removing these logs.

   *Resolution*: Fixed by the Customer's team.

7. Storage variables can be reordered by accident. There is a warning in `JLoanStorage.sol` which notifies about the danger of re-ordering. But if someone adds a variable to `GeneralParams` or `FeesParams` structs, it will shift storage, because structs are stored by value. Consider adding a warning to never reorder variables to `GeneralParams` and `FeesParams` structs.

   *Resolution*: Fixed by the Customer's team.

## Conclusion

According to the technical analysis contracts were manually reviewed against common development practices. The team has found some high-level and medium-level issues during the analysis and the report contains all necessary information related to them as well as recommendations for issues' elimination. Though, all critical issues were resolved by the Customer's team.