# Intern Coding Challenge: Log Parsing and Testing

## Scenario

You have been hired as a development testing specialist with a company that performs automotive testing. You have been given a log file that was the result of a recent test and you have been asked to write a program that can both parse and consolidate information in the log file. Even though we have asked our testing team to keep the log file free of any modification, testers have still written comments into the log file itself for future reference. We need your solution to be able to handle this data as well as any other discrepancies that may exist in this log file, or future log files that we use your program to parse.

## Technical Requirements

The log file consists of individual messages that are either being sent to the target or being received from the target. It is a requirement that the target responds within a certain timeframe in order to pass our tests. In other words, if an instrumentation attempt isn’t immediately followed by a response, a Denial of Service (DoS) failure is present. The way the DoS time is calculated is by taking the difference between the time that the original request was sent and when the target responded. Some examples from the log are shown below for verification.

Example:

Test case #5 – DoS time is 0:00:20.414

Test case #415 – DoS time is 0:00:20.405

Test case #1395 – DoS time is 0:00:06:070

The final product should be able to calculate the DoS time for every failed test case in the log, while handling any inconsistencies in the log itself. We will consider any DoS time that is within 100ms of the verified time an accurate result (refer to the example above for good verification times). Also, please think about how your program will output the results of the parsed log file and whether or not this is a persistent solution (i.e. outputting results to the console session within the program would not be a good way to record these results for later). The output portion of the assignment is purposely left open-ended.

## Other Requirements/Suggestions

* Must be written in a modern programming language of your choice (Python, Java, C# are some examples).
* Thorough testing of your code is highly recommended. We are looking for a very robust solution that has been tested thoroughly. Scripting acceptance tests to automate this verification process is encouraged, and unit testing of code is also welcome.
* Make sure your coded solution is clean and modular. Ask yourself: will someone else be able to read and understand the code that I write? Could my program be used over and over again, or be reused for another similar purpose?
* The final product should be defect-free, and you should be able to prove that with recorded results from your testing.
* DO NOT RUSH. In our opinion, a well thought-out and constructed program that has been thoroughly tested, but is not fully complete is better than a complete program that is overly complex or full of errors.