# ISP Assignment 2.3 - Learning from mistakes

My reference monitor had 3 mistakes that were caught by 5 people's attack cases.
The issues related to *race conditions* caused by improper locking, *incorrect order of raising exceptions* in case of issues, and incorrect understanding of the *writeat assignment spec*.

1. **Race Conditions**: I had not used locking within my reference monitor program during initial development. This led to attackcases exploiting my program by running multiple threads that did read, write, undo operations on a file in a loop. During some of the iterations of those loops, the state I maintained within my LPFile object became inconsistent after it was correctly checked, and the original Repy API operations were called with the inconsistent data. This led to errors being generated.
   For example, my writeat function does processing to ensure that the pending_data is correct before passing it to the original Repy writeat. But if undo is called after checking the data but before the actual call to original writeat, it destroys the data that is passed (changes it to None) and thus the original writeat throws an exception.
   This is not acceptable and if the state of the variables was locked between start and end of my writeat implementation, undo would not have executed in the middle and destroyed valid data. My new implementation takes care of locking.
   *Two attacks* did correct threading attacks to fail my reference monitor.

2. **Exception Ordering Error:** My writeat function did argument and state checking in the order of RepyArgumentError, SeekPastEndOfFileError and then FileClosedError. The actual Repy api diverges from this behavior and does checking in a different order - RepyArgumentError first, FileClosedError second, SeekPastEndOfFileError third. For example, if writeat is called on a closed file with offset past EOF, Repy API states that FileClosed error should be raised but my implementation raised SeekPastEndOfFileError error which is wrong. This difference in implementation led to unexpected exceptions being generated that were not caught by the testcase handlers and bubbled up to generate errors while running. Repy API clearly mentions that the order of exceptions mentioned in the docstring should be followed.
   Now my implementation has been corrected according to the API.
   *One attack* did this check and generated unexpected output from my ref monitor.

3. **Empty writeat flush:** I assumed that an empty writeat should not flush the contents of the buffer into file. I assumed this because some text editors that allow undo operation still allow undo after file has been written to disk. And the first undo operation leads to change of content and not just unsaving of the file.
   But according to the assignment specifications, this assumption is wrong. The assignment spec trades off an extra undo operation for the ability to have a flush command. The assignment spec (second point, first bullet) mentions that the changes can be considered permanent if a valid writeat operation is performed. An empty writeat is a valid writeat according to the Repy API, thus we should allow flushing of the buffer to the file in case of empty writeat. It has been corrected now within my ref monitor.
   *5 attack cases* did a check for this and failed due to my incorrect implementation.