Assignment 2 Part 3 Writeup

According to the current state of the attack matrix, no valid attack case that ran against my reference monitor has passed. However, I encountered 1 unique scenario:

In wrapper.r2py, there is a check: "***match = isinstance(args[index], expected_types)***" in line 460, which is supposed to check if each argument passed to a function matches the expected types. But, since bool is a subtype of int in python, this test seems to accept booleans even if expected_types does not contain <type 'bool'> . Therefore, even though the reference monitor does type checking for its args like in "***writeat": {"type": "func", "args": (str, (int, long)), "exceptions": Exception, "return": (int, type(None)), "target": LPFile.writeat}***", passing True/False as writeat()'s second argument fails to be caught during safety check.

This caused the testcase which passed True as writeat()'s second argument (offset) to enter my function without being blocked, and led to an erroneous result.

I have fixed this by adding a manual check to validate the type of the second argument to writeat() and readat(). This is not a problem with undo() or close() since they do not accept arguments.

When I wrote the reference monitor for part 1, I discovered a couple of errors/issues which I had fixed/resolved before submitting. They were:

1. Behavior of undo() after the file is closed - undo() after the file is closed should return without doing anything.
2. Behavior of readat() when buffer is non-empty - read only the contents present in the file without considering the contents stored in the buffer.
3. Behavior of writeat() to an offset that's not valid in file, but valid when considering the buffer - the file size needs to be maintained as part of class state, and validity of offset is determined based on current file size as well as anticipated file size when considering pending data and offset in buffer. The invariant in my implementation is that the 'filesize' variable always reflects the current size of the open file. It is updated only when data is actually written to it.
4. Error handling for invalid arguments vs error handling for runtime errors - The reference monitor should handle invalid argument errors (like negative offset, or wrong variable type) before it tries to acquire a lock (since in these cases it can be known with certainty that the call will fail). Whereas runtime error handling like checking if a given offset is valid and whether the file is closed, must always be checked after acquiring the lock to prevent data races.
5. Acquiring and releasing locks correctly - locks must always be released when leaving a critical section, regardless of whether an exception was raised (using *finally*).