Justin Rivera (jnr8342)

Assignment 2.3 Write-up

After running all other attack cases against my reference monitor, there were a few observations I made regarding vulnerabilities in the monitor I implemented.

Firstly, I initially implemented locks into my reference monitor to protect against multithreading attacks. For the `readat()`, `writeat()`, `undo()`, and `close()` operations, a lock was acquired before attempting the operation and released at the end of the operation. However, there was a vulnerability in the `close()` operation. For the `readat()` and `writeat()`, my reference monitor handled operation invokes that raised an exception. This was handled by wrapping the potentially erroneous code in a try-catch block, releasing the lock in the event of an exception, and then raising the exception. This functionality was not implemented for the `close()` operation. I failed to include a try-catch, so the lock would be acquired and never released if the file was already closed and threw a `FileClosedError`. If any operation were called following the erroneous `close()` call, it would result in deadlock and cause a failure. This was fixed by implementing a try-catch block in the `close()` operation in the same manner that was used for implementing the `readat()` and `writeat()`.

Another vulnerability was found in my `writeat()` operation. According to the RepyV2 API, there is a precedence to which errors should be raised when `writeat()` is invoked. `RepyArgumentError` has the highest precedence, followed by `FileClosedError`, and finally `SeekPastEndOfFileError`. However, my `writeat()` implementation did not follow this order. Instead, `FileClosedError` had the highest precedence, then `SeekPastEndOfFileError`, and then `RepyArgumentError`. This was corrected by changing the order in which erroneous cases were handled in my `writeat()` implementation, such that the new order would follow the RepyV2 API.