INTERNET SECURITY AND PRIVACY | Assignment 2.3 | Shiv Thaker | st5073@nyu.edu

My rationale behind vulnerabilities and how I fixed them?
My submission to assignment 2.1 implemented the undo functionality in the reference monitor. As per the specifications, the undo will revert the changes of the last writeat operation. A writeat will not immediately write its changes into the file, the changes will only be permanent after either a new writeat operation is executed or if the file is closed. In these cases, the pending data will be written at the pending offset, these variables will be stored with us in buffers. I did not handle any exception on my own, and I did not pass any exception. As repy was raising the exceptions on its own, I did not feel the need to raise the exception.

After assignment 2.2, I reviewed all the attack files and their outcomes on my reference monitor (the code I submitted in assignment 2.1). I noticed that the first mistake I made was that I did not take into account multiple threads reading and writing into a single file. This can raise a lot of inconsistencies. I implemented a thread lock using the below code:

```
self.thread_lock = createlock()
```

And I put in all the thread sensitive functions such as readat, writeat, undo, and close in between these locks. One such example is shown below:

```
self.thread_lock.acquire(True)
try:
        ……
        “”” function code in between ”””
        ……
finally:
        self.thread_lock.release()
```

I implemented my own error handling, as I felt the error handling implemented by repy was not enough. I implemented error handling for scenarios in writing data into a file, in case the user uses a negative offset, as shown below:

```
if offset < 0:
        raise RepyArgumentError("Read offset cannot be negative.")
```

And in case the user attempts to write beyond the end of the file, as shown below:

```
if offset > self.file_length:
        raise SeekPastEndOfFileError("You cannot read data past the EOF.")
```

I implemented the mechanism to update the current length of the file. This will enable us to ensure that the user will not be able to write data into the file beyond the end of the file. Shown below:

```
self.file_length = len(self.LPfile.readat(None, 0))
```

I did not raise an exception in case the file is closed multiple times. This is not the correct behavior. In this submission, if the user tries to close the file multiple times, I raise an error message as shown below:

```
if file is already closed:
        raise FileClosedError("File is already closed.")
```

My final submission catches all the possible errors, and it does not produce any output. I have turned in my reference monitor as reference_monitor_st5073.r2py.