A vulnerability I found from the tests was my lack of locks which made it unable to deal with threading causing race conditions and indeterminate outputs. Most of the vulnerabilities I found based off of the tests were that I did not apply the same logic and testing to readat, undo, and close as I did for writeat. For close I did not track whether the file had been closed already meaning that it could close twice causing a FileClosedError. I also did not check whether the byte size in readat was larger than the actual file size, meaning that it could try and read past the end of file. I also did not check if the file was closed when trying to read or undo. I also did not check if the readat byte value was a valid argument by checking if it is negative. I also forgot to check if my pending_offset was None before using it to check if the new offset is past the max possible offset causing an error of adding int and None.

I addressed the lock vulnerability by adding an attribute called self.lock which is a lock object. I also then added a self.lock.acquire before every method reading or writing the critical section and used a try for the actual code and finally to close the lock using self.lock.release. As for the file being able to be closed while already closed I added a self.closed variable to check whether the file is closed and will become false when initialized and then true when the file is closed and false when the file is opened in the open method. I added a check before writeat, readat, and undo to check if the file is closed and raised an exception if it was. I also added another condition to the if statement before the offset and max offset check. I also raised a SeekPastEndOfFileError if the writeat offset was past the max offset and also raised a RepyArgumentError if the offset was negative now throwing exceptions when the input is not valid