Tom Zhang
HW2.3

**Vulnerabilities Fixed**

1) Incorrect file size set in '__init__'
   a) In my '__init__' method, I always set the file size to 0 if the parameter 'create' was true. However, in cases where the file already exists and contains data, the file size should not be 0. Thus, when opening a file using 'create=True', any writes with an offset greater than 0 would incorrectly result in a SeekPastEndOfFileError.
   b) To fix this, I set the file size to the number of bytes returned when reading the file during initialization. Thus, whether a new file was created or an already existing file was opened, its file size would be set to the proper value.
2) 'readat' when 'undo' method is called
   a) In my 'undo' method, I called 'readat' to get the file size prior to the previous write. However, when 'undo' was called after the file was closed, it incorrectly raised a 'Fatal Error'.
   b) To solve this, I removed the 'readat' call from my 'undo' method and replaced it with an instance variable named 'old_filesize'. As implied, this would be the file size prior to the latest write. Upon initialization, 'old_filesize' and 'filesize' would have the same value. During every write, I would copy 'filesize' to 'old_filesize' before storing the new file size.
3) 'bool' types passed into 'writeat' as an int
   a) Since booleans are a subclass of integers in Python 2, they could be passed into 'writeat' without being detected, which oftentimes resulted in errors during write commits.
   b) In response, I added explicit argument checks for booleans to the 'writeat' method. If a boolean was detected, the method would throw a RepyArgumentError.
4) Invalid file state from parallel read or write calls in a multithreaded attack
   a) I did not implement any locks in my reference monitor, so any multithreaded attack could perform file IO in parallel, resulting in an inconsistent file state.
   b) To remedy this, I created read and write locks and called the 'acquire' and 'release' methods before the 'readat', 'writeat', 'undo', and 'close' operations. Doing so forces the operations to execute sequentially, ensuring a consistent file state.