Jarred Carter
Professor Justin Cappos
GY 6813
31 October 2023

Modifications to Reference Monitor After Programmed Attacks

This simple yet challenging assignment was a fascinating exercise in persistence and resilience. In fixing errors, the addition of code to remediate most attacks resulted in the depreciation of the undo functionality implemented in Assignment 2.1, requiring the void of all changes and the rewriting of the reference monitor altogether. Upon further examination, the number changes implemented below were simple, yet it did not occur to me to implement them all at first. The following code additions brought down the number of attacks from a number over 370 to under 45 within a matter of days:

1. **Addition of Locks:** `self.lock = createlock()`
   The first addition to the code was the lock feature, which functions similar to Python's. Code additions here, which include the acquisition of a lock before try blocks and/or other necessary calls and a release at the end of a block, function by allowing a thread to proceed, while other threads attempting to acquire the same lock will be blocked until the lock is released. This prevents deadlocks and race conditions while simultaneously controlling race conditions.
2. **Check for File Closure Before Write:** `if self.closed`
   A check to see if a file is closed before writing has been included in `writeat()` and `undo()`, and will raise a FileClosedError in `writeat()` and a log in `undo()`, as an attack case prevented the usage of FileClosedError in the `undo()` function. This is tracked with the self.closed variable defined in de __init__, which is set to False at the beginning of the program and is later changed to True in `close()` before releasing the lock.
3. **Check for Offset Larger Than File:** `if offset > self.len`
   An offset that extends past the end of the file caused unexpected behavior at the time of writing, hence a check to see if the offset was larger than the file itself eliminated this error. A SeekPastEndOfFileError is raised and displays the message "Offset extends past the EOF."
4. **Check for Negative Offset:** `if offset < 0`
   A negative offset threw similar errors as in (3). A RepyArgumentError is raised and displays the message "Cannot write with a negative offset."
5. **Check for Offset Greater Than Current File Size:** `if offset + len(data) > self.len`
   While running attack cases en masse against the reference monitor, an issue was encountered where, with one test case, a second write with an offset of 1 which followed a write with an offset of 0 threw an error. Nothing was written to the file in this case, whereas for other test cases, data was written. This check sets self.len equal to the size of the offset plus the length of the data the user is attempting to write, thus eliminating the issue.