Maya Humston
Professor Cappos
Computer Security
2 November 2023

Assignment 2.3

Part 1 of this assignment was very interesting but unfortunately very confusing for me. I vastly misunderstood the instructions (my own fault) and ended up with a very poor security system – I got VERY close to understanding, but I thought that "your security monitor must produce no output" meant that any exceptions thrown should NOT be printed to the monitor, so added exception handling that kept them from being thrown properly. This meant most of the attackcases bypassed my layer. Oops! I enjoyed the attacking more. It helped me better understand the assignment. I've never done Python multithreading before but I had to try, and I think I *almost* got a working multithreaded attack. I understood when exceptions should be thrown and made attacks accordingly.

Having done this, I knew my own flaws better now. Firstly of course, I fixed my exception handling so that it threw exceptions properly instead of stifling them. This took care of the majority of my security flaws. I also fixed the math in my eof calculation. I think the way I set it up had an edge case that produced bad math.

During part 1, these were my priorities:

- Throw exceptions for bad parameters
- Store the pending information in a buffer
- Write from the buffer when 1) the file closed and 2) another valid writeat was called
- Call readat once during init to track initial eof
- Track eof after every writeat call
- Implement an undo function

Having done this + attacks, I knew my own flaws better now. Firstly of course, I fixed my exception handling so that it threw exceptions properly instead of stifling them. This took care of the majority of my security flaws. I also fixed the math in my eof calculation. I think the way I set it up had an edge case that produced bad math.

I also added locks. I have never done locking/multithreading before so that was a neat thing to learn about. I hope I did it right – I added a lock to the self instead of to the self.LPfile but I'm not sure if that makes a difference. I know enough about critical sections to know that ANY code that accesses shared memory is a critical section that should be locked. I interpreted this to mean that writing, reading, undoing, AND closing the file should all be implemented with locks, because all of these operations edit at least one variable stored in the "self". I've seen other students lock only for reading and writing and I don't know if undoing should use locks, because they don't edit the LPfile itself.

I also made sure to better use try/except blocks and tried to ensure that I never exit a function without unlocking the lock, even if an exception was thrown.

After this, most of the attack cases were handled. There was only one that I couldn't figure out: rsh attack case 10. I know there were three threads, one for reading, one for writing, and the last one closed the file. In all my runs, the third thread closed before read could occur, so read threw a FileClosedError. I don't know how to handle this case or what the correct implementation is. My only guess is that I should not use a lock for either the read or close function, but I don't think this is true, so I'm curious what the fix should be.