Bradley Cushing
bsc6146@nyu.edu
Assignment 2, Part 3
2023-10-28

After submitting my reference monitor there were a few vulnerabilities found. Most attacks were around two cases in my reference monitor that I either missed or implemented incorrectly. The first two vulnerabilities were found to be related to my implementation tracking the offset for changes. While writing my own tests I realized I had forgotten to set the offset for files when reading them in based on the content that was already in the file. This was a simple mistake I made that I forgot to test when writing my code. I wrote a test to exploit this vulnerability in other reference monitors including my own. The second offset bug in my code was just a mistake I made while refactoring things. I was mistakenly updating the maximum offset I was tracking in the reference monitor before calculating and setting the offset delta. This caused the offset delta value to be wrong and there was a flaw in my early tests that didn’t catch this as well. I just needed to set the offset delta before the maximum offset and that fixed the issue.

Mutex locks and threads were another interesting area of concern. In my original reference monitor I didn’t include any locks for critical sections. I don’t use any global variables but I am obviously accessing file handles which critical sections and should only be able to be accessed when at a time when multiple threads are being used. I believe there are problems with almost all if not all of the attacks that are written using threads but I still went ahead and added locks to my reference monitor to prevent any legitimate attacks that are correctly using threads. This fix involved creating a lock in the constructor, acquiring the lock before running any code for a given method, wrapping all code to be executed in a try block, and then in a finally block releasing the lock. Releasing the lock in a finally block is important because the lock should be released when the code is executed successfully but also when it is exited unsuccessfully as to not cause a lock in a thread to be held indefinitely and cause a deadlock. Lastly, there were two tests that were timing out past the allowed 10 seconds for an arbitrary attack but when investigating those cases further the timeout was due to the fact that these tests were incorrectly meant to take a lot of time. One of them was broken and eventually logged a result regardless of the correctness of the reference monitor. The other test only slept for a total of 9 seconds but added to the rest of the test it took longer than 10 seconds.

Reflecting on the exercise I thought it was really good and I definitely learned some lessons that will stay with me due to the hands-on nature of this learning experience. I was definitely surprised at how such a relatively simple extension of functionality resulted in the need to be concerned with so many different things. When I first saw the exercise I didn’t think there would be too much to handle after getting the basic cases working but I kept discovering possible attacks when writing my reference monitor. I was a bit disappointed that I didn’t prevent all attacks. I was definitely thinking about those things when writing my code and tests for the first part of the exercise but I had some small gaps in my tests and wasn’t thorough enough. I also haven’t worked much with multi-threading before so this was a good experience to understand possible attacks under those circumstances and actually implement locks to prevent them.