Exercise 4:
Denial of Service

Contents

1 Tasks

1.1 Writing the Program

It should first be noted that in order to fully understand the below explanation, the code located in the files **dos.py** and **tls.py** should be opened and read in parallel.

For this exercise, we are going to use the *OpenSSL* Python library (code found on *Github* [1]) together with the *socket* library. This will enable us to skip the step of creating our own *ClientHello* packet.

As explained in the exercise, all threads should make a connection to the server first and start with the handshakes after each of them has successfully connected. We can therefore create a class named **TLSConnection**, in which we will initiate a connection in the constructor method and create a separate method named *do_handshake()*, which will initiate the handshakes after all the threads have made a connection.

In order to wait for all the other threads, we will use a *Barrier* object. As descrbied by the Python Docs [2]: *This class provides a simple synchronization primitive for use by a fixed number of threads that need to wait for each other. Each of the threads tries to pass the barrier by calling the wait() method and will block until all of the threads have made their wait() calls. At this point, the threads are released simultaneously.*

For our purposes, we will set the barrier value to 2001 and spawn 2000 threads. Each time a thread enters the *do_handshake()* function and executes the *barrier.wait()* function, it will increase the barrier counter, up until it reaches 2000. When all 2000 threads are spawn, we will call *barrier.wait()* one more time from the *main()* function and try to make all 2000 handshakes simultaneously.

After the handshakes are initiated by calling the *send()* method on each connection, we can count all successful and failed handshakes by using a *try-except* statement. If the handshake fails, it will execute the *except* part, thus increasing the *failed* counter and exiting. Else, it will break out of the *try-except* statement and increase the *success* counter.

In the main function, we will check for failed handshakes every 0.1 seconds. If we encounter one or more, we will save the elapsed time from the beginning of the handshakes until that moment in a variable. We will also save the number of successful handshakes into another variable. This information will later be used to calculate the number of successfuly handshakes per second before the server starts to reject further handshakes.

1.2 Testing

In order to test the program, we would first need to increase the file size limit to, e.q. 4096. This would allow us to spawn more than 1000 threads:

```
1 ulimit -Sn 4096
```

Listing 1: Increase file size limit

In order to check if it is working correctly, we can execute it a couple of times and see if the output is continuous. We can then calculate the average time elapsed before a failed handshake, as well as the average number of handshakes per second before the first failed handshake. The execution results are shown in Table 1. The last row of the table represents the average of all results.

1.3 Results

#	Successful handshakes	Time Elapsed	Handshakes per Second
1	1,569	3.7 seconds	424
2	1,695	3.8 seconds	446
3	1,721	3.7 seconds	465
4	1,649	3.9 seconds	423
5	1,517	3.7 seconds	410
6	1,707	3.7 seconds	461
7	1,765	3.8 seconds	464
8	1,590	3.7 seconds	430
9	1,662	3.7 seconds	449
10	1,683	3.8 seconds	443
	≈1656	**≈3.7**	**≈445**

Table 1: Results

2 Requirements

No requirements needed, everything should already be installed on *Ubuntu 20.04 LTS*.

References

1. https://gist.github.com/shanemhansen/3853468
2. https://docs.python.org/3/library/threading.html