Producer-Consumer Code (code source: https://techmonger.github.io/55/producer-consumer-python/)

Ways to improve security:

1) Instantiate a Producer first (before instantiating consumer threads and starting them)

At the moment, one has to understand the specific function “Queue.get()” and its default values (= point 2) so that he/she can understand why the program runs. Another option would be to start the program in instantiating a producer and then calling the consumer threads. Why this can be more secure? It simply makes the code more readable and it is in case of an attack also faster reproducible.

2) Queue == empty check for the consumer class at the beginning:

Because at the moment the code works because the default option of the Queue.get() function is “block = True” and “timeout=None” which simply blocks the get request until an item is available (see Python Documentation). Why this can be insecure? It might not be but including before the “q.get()” command also a check whether the q is empty makes the code more redundant. The result would be that running the code would need more time (because of 1 more conditional jump) but security would also be increased (because the code doesn’t depend too much on the “get()” function of the Queue library).

3) Queue == full check for the producer class at the beginning:

As seen with Queue.get(), the same is true for Queue.put() which also has as default values “block = True” and “timeout=None” which this time allow to put entries into the Queue as long there is still space into it and if it is not, then it still waits until there is

again space available. In changing the whole structure of the code as described in point 1 and point 2, the “Queue == full” check could be included and would again lead to more redundancy (see explanation of point 2, here idem).

Here is the suggested code:

```
from threading import Thread
from Queue import Queue

q = Queue()
final_results = []

def producer():
    for i in range(100):
        if not q.full():
            q.put(i)


def consumer():
    while True:
        if not q.empty():
            number = q.get()
            result = (number, number**2)
            final_results.append(result)
            q.task_done()
```

```
producer()


for i in range(5):
    t = Thread(target=consumer)
    t.daemon = True
    t.start()


q.join()


print (final_results)
```