1 Meltdown Attack Lab

Notice that **this attack is particularly dependent from the actual configuration of your pc** and its load. **Even recharging the laptop or connect external monitors affects result A LOT!**

1.1 Task 1

In this task we have to find a good threshold to distinguish between cached and not cached data. After a cople of tries like the one presented in the following, we set the threshold to `200`, but the value could vary for your machine.

```
seed@VM:~/.../scripts$ ./CacheTime
Access time for array[0*4096]: 1504 CPU cycles
Access time for array[1*4096]: 258 CPU cycles
Access time for array[2*4096]: 298 CPU cycles
Access time for array[3*4096]: 138 CPU cycles // cached
Access time for array[4*4096]: 420 CPU cycles
Access time for array[5*4096]: 454 CPU cycles
Access time for array[6*4096]: 456 CPU cycles
Access time for array[7*4096]: 134 CPU cycles // cached
Access time for array[8*4096]: 290 CPU cycles
Access time for array[9*4096]: 268 CPU cycles
```

It is important to notice that other software running in the same machine can have an impact on the access time: therefore, on different runs, there may be big differences. Try to find a good threshold but take into considerations that the following attacks could fail from time to time.

2 Task 2

If you did everything correctly, you have to see something like:

```
seed@VM:~/.../scripts$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

We find it in about 70% of the 20 run of the application. However, it is really variable, so values above 50% are great. Furthermore, it you have bad results, try to modify the threshold and rememeber to recompile your program.

3 Task 3

When you comment the `$` line, you avoid to flush the value of `size`. This means that the `victim()` function is way faster to evaluate the if statement and therefore the time window in which the secret is in the cache is way shorter then before. Thus, hit the secret is almost impossible.

The likelihood to hit the secret is almost impossible since the training to take the "true" branch of the if statement is never taken. On the countrary, it gets used to take the false branch and the speculative execution is not made anymore.

4 Task 4

Check `SpectreAttack.c` in `Solution` folder.

5 Task 5

Check `SpectreAttackImproved.c` in `Solution` folder.

- fix at line 96 (`max` instead of `scores[max]`): `if(max < scores[i])` rather then `if(scores[max] < scores[i])`
- On Ubuntu 16 the attack is working somehow even without that line, but it losts quite a lot of chars of the final result. Instead, with that line, the string is almost always fully reconstructed.
- On our configuration, ranging from 1 to 500 there are no changes on the success rate (apart from the natural delay in execution).

6 Task 6

Check `SpectreAttackImproved.c` in `Solution` folder.