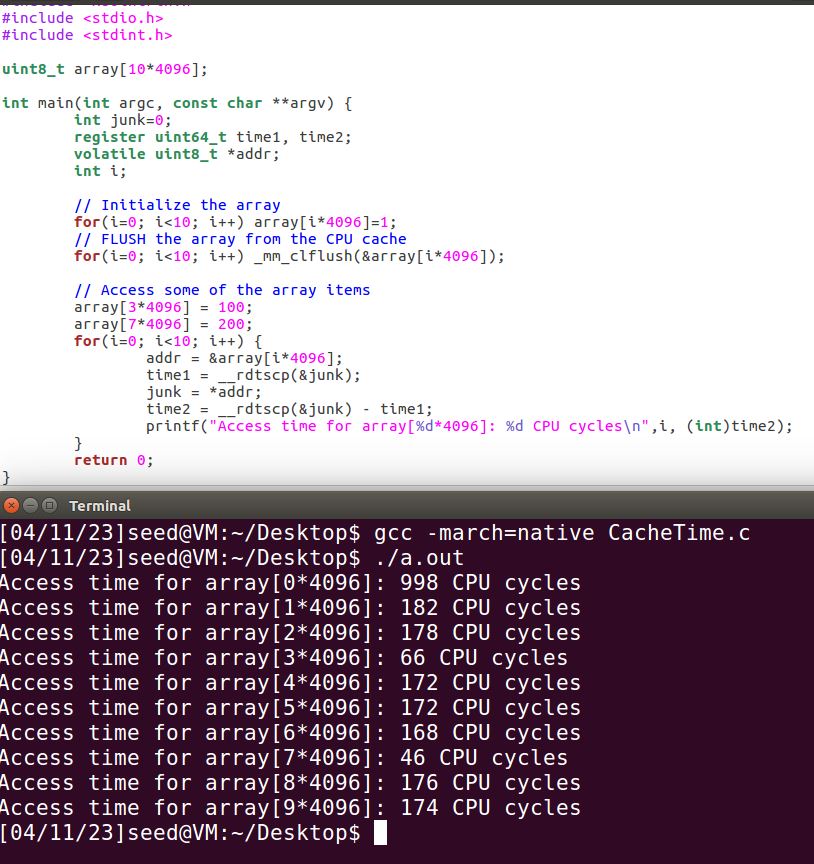
Adam Jarvis 4/11/2023

Project 5: Meltdown Attack

In this project, I was tasked with completing tasks 1 through 6 of the Meltdown Attack project offered by SEED Labs. The Meltdown exploits critical vulnerabilities that exist in many modern processors, especially Intel and ARM processors. Overall, this exploit allows programs at a user-level to read data stored in the kernel memory.

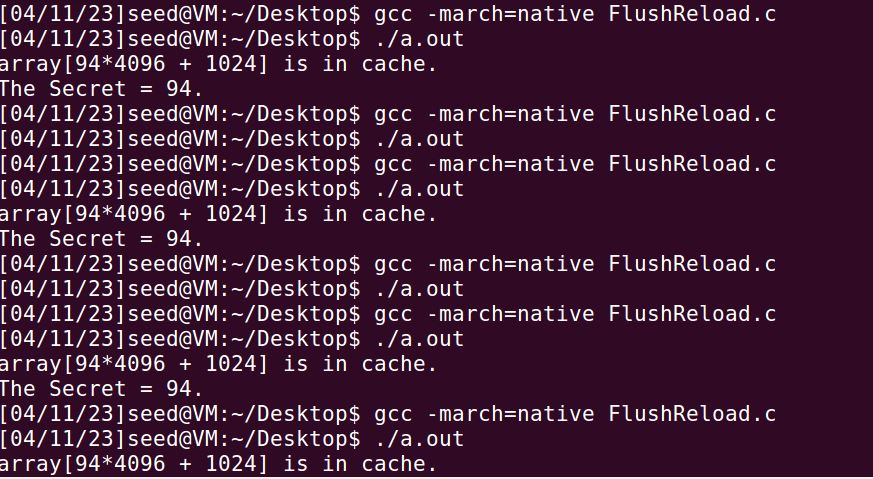
**Task 1**

For task 1, I used SEED Lab’s starter code for CacheTime.c and ran it using “gcc -march=native CacheTime.c”:



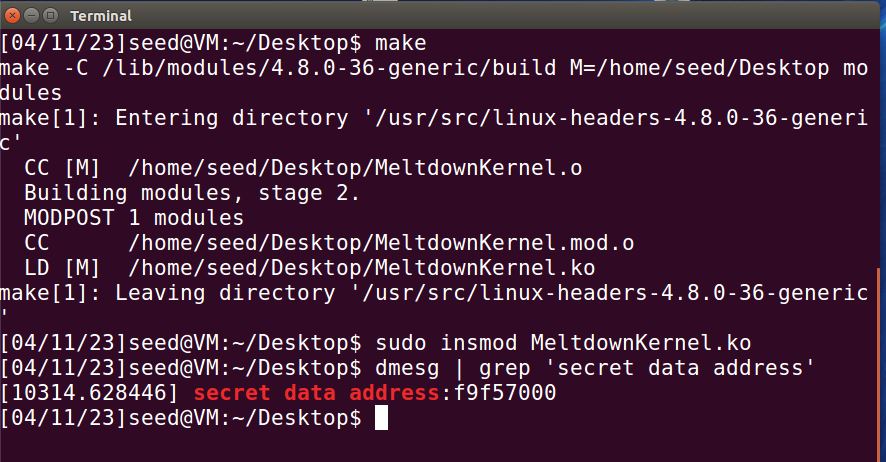
From the picture, we can see that the access time for array[3\*4096] and array [7\*4096] are significantly lower than other access times for the same array at different indices. This is due to the CPU caching the pages for array[3\*4096] and array [7\*4096] after setting the values to 100 and 200 respectively. Then, in the for loop when reading the array, the cache is hit for array[3\*4096] and array [7\*4096] and is missed for the rest, which leads to significantly lower access times for array[3\*4096] and array [7\*4096].

**Task 2**

For task 2, I used SEED Lab’s starter code for FlushReload.c and ran it using “gcc -march=native FlushReload.c” several times:

From the above picture, we can see it prints out the secret a little over half of the time. I also tried changing the value for CACHE\_HIT\_THRESHOLD and noticed little change.

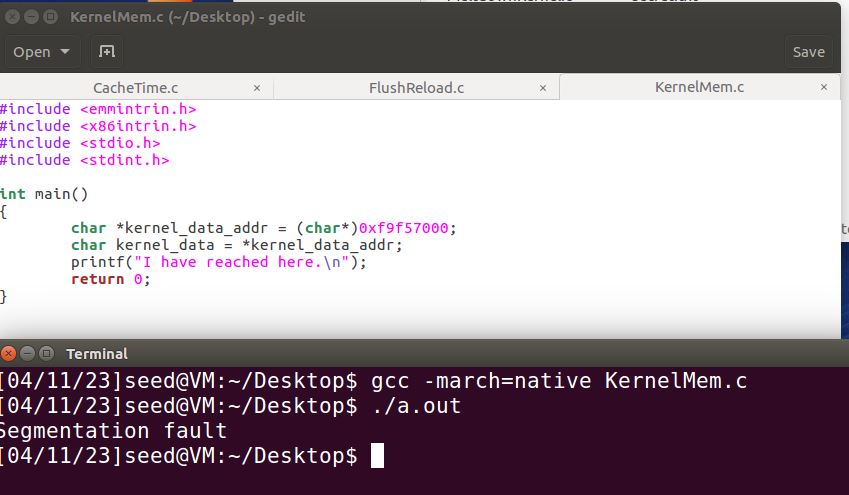
**Task 3**

For task 3, I downloaded MeltdownKernel.c and the Makefile from SEED Lab’s website and used a shared volume to copy the files from my host machine to the Ubuntu virtual machine. I then started a terminal at the same directory as these files and ran the following commands:

From the above picture, we can see the secret address is 0xf9f57000.

**Task 4**

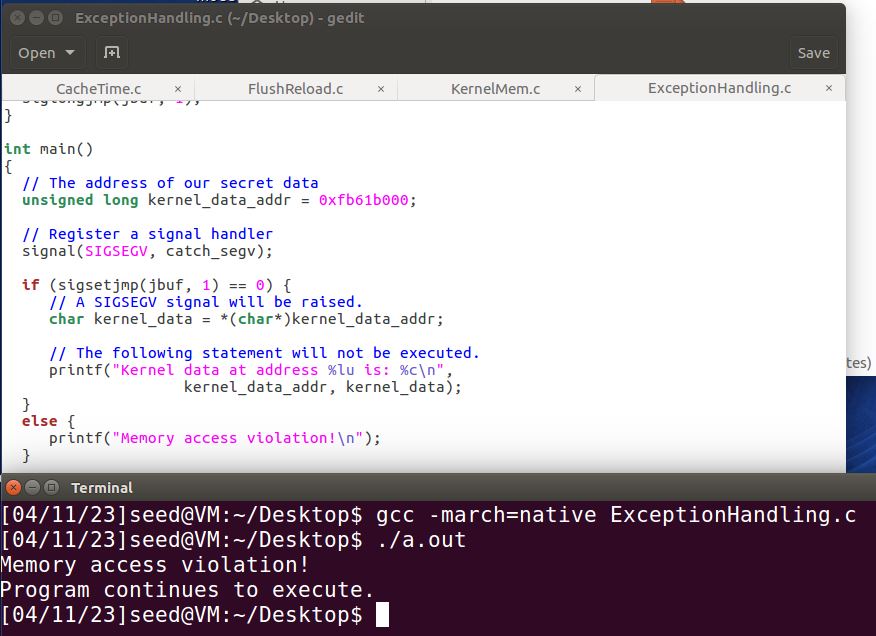
For task 4, I was tasked with experimenting if I can directly get the secret from the above address. I created KernelMem.c using SEED Lab’s starter code and made several modifications. I then ran it using the following commands:



From the above picture, we can see the program crashes with a “Segmentation Fault” error. Segmentation fault errors primarily occur when any program tries to access a part of memory that cannot be accessed. So, as a result, we can’t directly get the secret from the address.

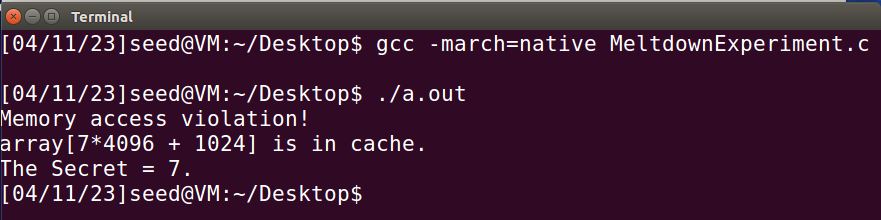
**Task 5**

For task 5, I was tasked with using SEED Lab’s starter code ExceptionHandling.c to see if we can handle the error without completely crashing the program. I ran the program using the following commands:



From the above picture, it’s evident that the error still occurs but the program continues to execute. We can use this technique and knowledge in the next task to create a final Meltdown exploit.

**Task 6**

In the final task, I used SEED Lab’s starter code and code from previous tasks to put together MeltdownExpirement.c. After entering the address I got from task 3, I ran the program using the final commands:

From the above picture, it’s obvious that even though an error occurs, the program still recognizes the array in cache and prints out the value of the secret variable.

**Conclusion**

In conclusion, this project provided an overview of the Meltdown Attack, a critical vulnerability that affects modern processors. Through the completion of tasks 1 through 6, I was able to successfully analyze and experiment with various techniques to return secret data from kernel memory. Task 1 allowed me to understand how CPU caching can be exploited, while task 2 demonstrated the use of Flush+Reload to extract data from the cache. Task 3 helped me identify the physical memory address of the kernel data, and task 4 highlighted the limitations of directly accessing kernel memory. In task 5, I learned how to handle the segmentation fault error and continue program execution. Finally, task 6 enabled me to use the knowledge gained from previous tasks to create a functional Meltdown exploit that extracts sensitive data from kernel memory using CPU caching.