CYSE 211

Mahish Mahendarkar

7/14/2021

Meltdown Attack Lab 6

Lab 6 purpose:

The purpose of this lab is to carry out a CPU based meltdown attack. The attack is broken across several tasks and the main goal is to be able to print out secret data stored inside the kernel. The steps of the attack are as follows: Meltdown attack, Side channel attack, CPU Caching, Out-of-order execution inside CPU microarchitecture, Kernel memory protection in operating system, Kernel module.

---------------------------------------------------------------------------------------------------------------------

Pre-Lab configuration settings:

**Download Files:**

This screenshot shows the commands showing our proper file setup.

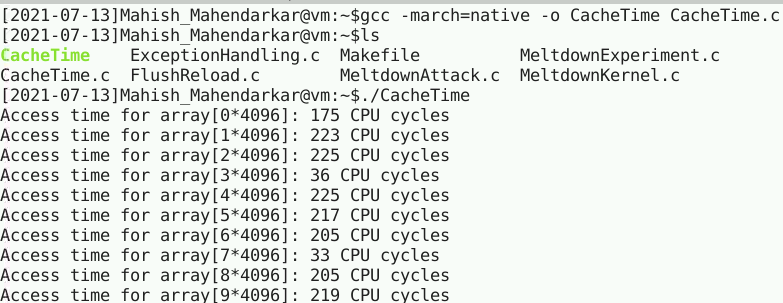


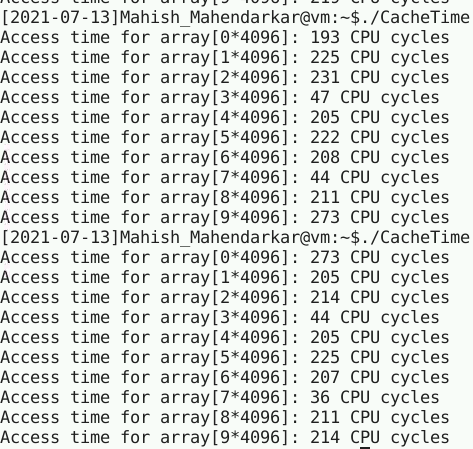
Lab tasks:

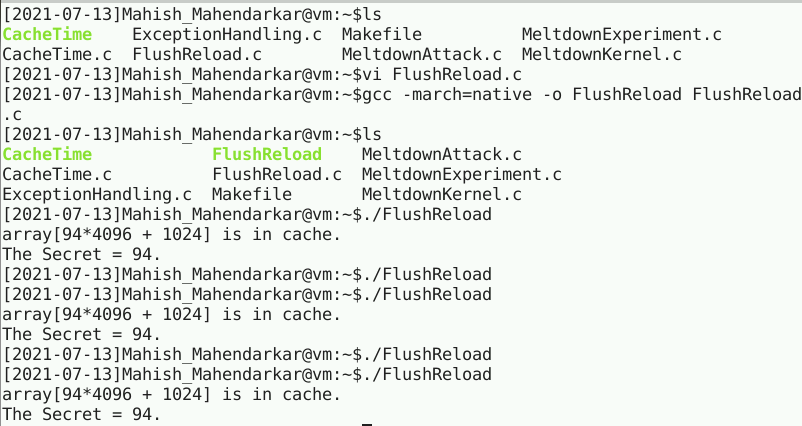
**Task 1 and 2: Side Channel attack via CPU Cache**

These tasks are fairly straightforward compile and execute actions based on the provided files. The first task compiles and displays full CPU information in the output. The second task runs a Flush and Reload program that is able to print the secret value being accessed which is the main part of a Meltdown attack.

Observations: The first file is compiled and run a few times. An important note is that the array of 7\*4096 always has the smallest CPU cycles value. Even after 3 runs the 7th cache block is consistently the ‘quickest’. This is an important inclusion in the subsequent calculations and files which have already been created. The next step is to compile and run the [FlushReload.c] file. This essentially clears the cache and reloads the cache based on the victim accessing the secret data. Then the program reloads the entire array and tracks the time taken to reload each entry. The time taken is analyzed the shortest time for an entry means that the entry has been pre-loaded. This entry could only be preloaded because the victim had accessed solely that array entry after the cache had been cleared out. The [FlushRead] is successful occasionally due to the nature of the system we are running this in.







**Task 3: Place Secret Data into Kernel**

These tasks involve placing the found Secret Data into the kernel. The two key parts are knowing the address of the secret data and then caching this data. There is a given Makefile to address the latter part.

Observations: The first part involves running the given commands as shown in screenshot 1. Initially I got an error as shown towards the bottom. The error essentially was telling me that my [insmod] command could not be run because the file exists. I was not sure what this meant so I restarted the lab from the beginning. I deleted all my files including the [MeltdownKernel.ko] and started from task 1. The last screenshot shows me running into the same error again. I decided to just try the [grep] command and it shows an address location for the secret data. I am assuming the first time I ran the [insmod] command it did in fact insert the module into the [MeltdownKernel.ko] file and clearly there is a address outputted which would not be possible if the prior command failed.

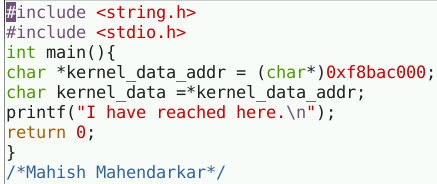


**Task 4/5: Access Kernel memory from User Space/** **Out-of-Order Execution by CPU**

This task is testing out a UserAccess program meant to directly read the Kernel Memory data and hopefully this can be used to read the secret data. The first part involves the creation of a file which contains a pointer to the secret data address and then printing out a successful statement. This should initially not work because privileged data can not be accessed by a regular user, this then should throw an error in execution. Then when an Exception is created to address this error the exception executable should allow us to violate the rules of memory in place.

Observations: The first part involves the creation of [UserAccess.c] and its subsequent compilation. This should not be successful as we are just a regular user and kernel memory is protected data. This is shown in screenshot 2.

(Start of task 5) Screenshot 3 shows that when the provided [ExceptionHandling.c] is compiled there is no segmentation fault and the program can continue to execute. This given file details new code to run when the initial error occurs, there is now an exception being handled and this allows for a success.



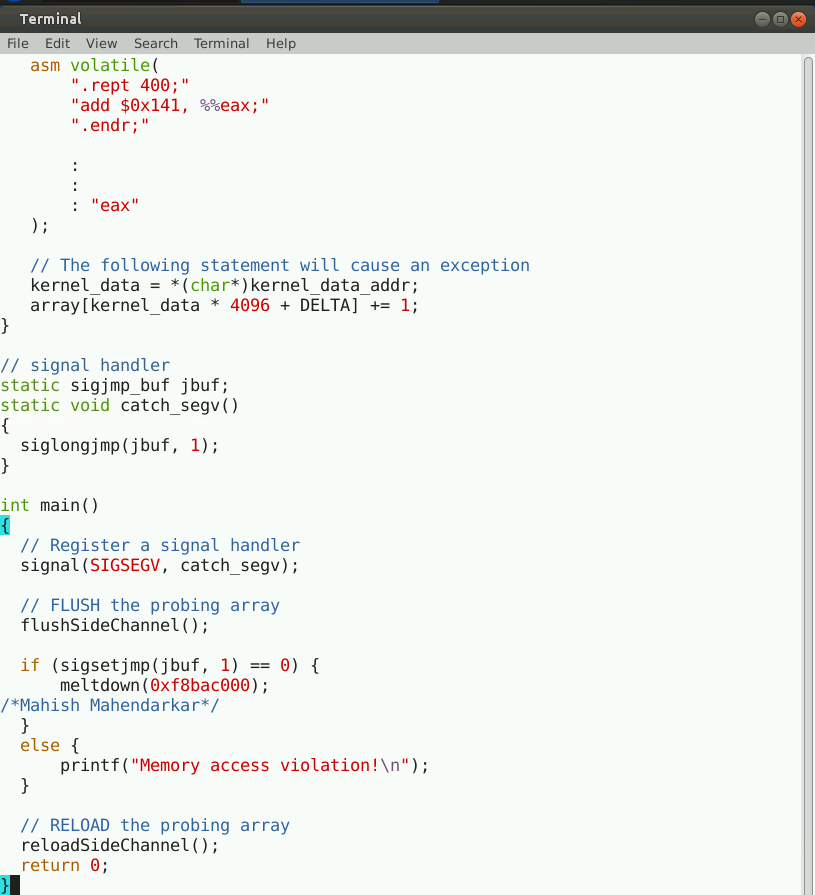




**Task 6: Out-of-Order Execution by CPU**

These tasks involve placing the found Secret address into the provided Meltdown Experiment program. This program is meant to cause an exception which leads to a coordinated out of order execution to get the cache setup for the later part of the attack.

Observations: The first part involves adding the address for the secret data as a parameter into the meltdown function. Then this file is compiled with special predefined permissions as shown in screenshot 2. When I initially had compiled this [MeltdownExperiment.c] file I had not compiled it with any special permissions and I ran into an error. This troubleshooting process took around 2 hours and I eventually thought to compile it with the given format of [gcc -march=native -o myprog myprog.c].

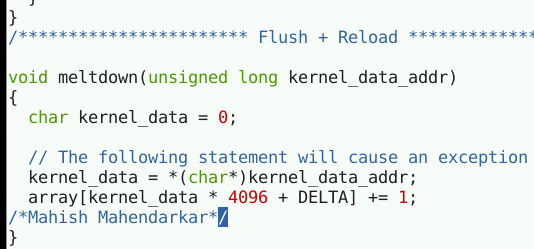


**Task 7: The Basic Meltdown Attack**

These tasks involve carrying out the full Meltdown Attack. The previous tasks have set us up properly and a key part is establishing the out of order execution ‘order’. The attack is a version of a race condition attack and the 3 parts below will show the successful completion of the attack.

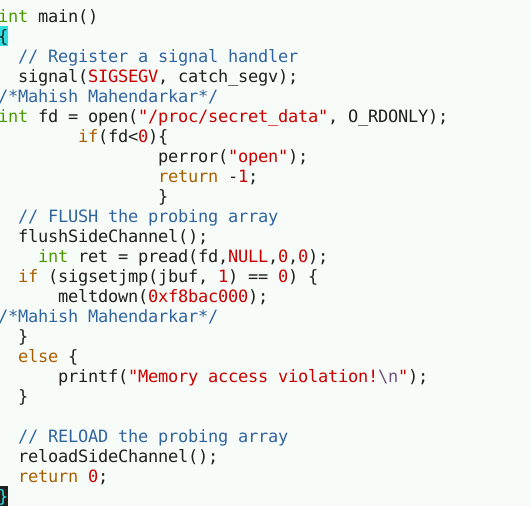
**7.1 – Modify MeltdownExperiment.c array[7 \* 4096 + DELTA] to array[kernel\_data \* 4096 + DELTA]**

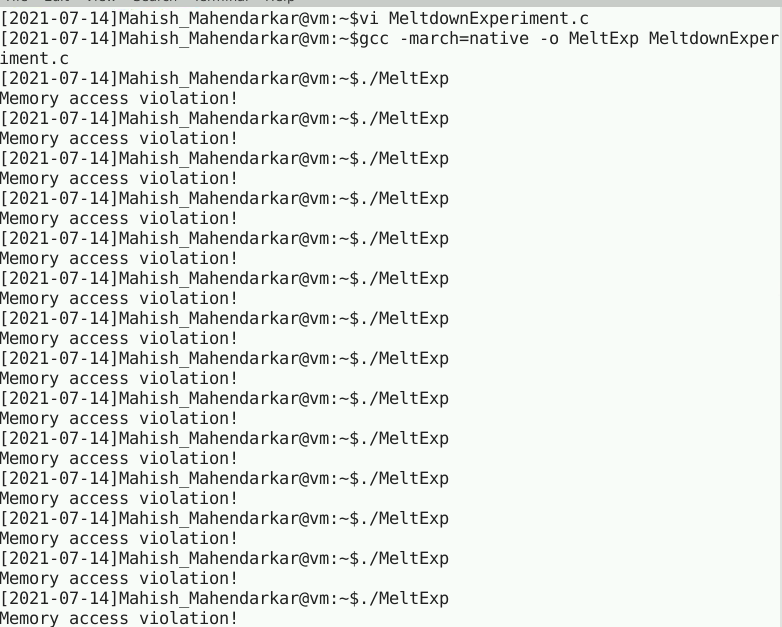
Observations: The first part involves changing the way the address is found from harcoding to a variable based method. This change results in unsuccessful program execution, this is explained in screenshot 2.

****

****

**7.2 - Improve the Attack by Getting the Secret Data Cached**Observations: The first part involves changing the [MeltdownExperiment.c]. The code is given to us but the exact location and insertion order are not immediately clear. I reread the instructions and implemented the code with the specifications of the attack order. This was able to compile successfully as a failure! I have not been able to successfully execute with full program being run since the first time I did in this lab.





**7.3** **Using Assembly Code to Trigger Meltdown**

Observations: The first part involves changing the [MeltdownExperiment.c]. The code is given to us already and the function being referenced now is [meltdown\_asm] not [meltdown]. The number of loops was initially 400 and it was unsuccessful. I changed the loops to 700 and ran the executable 3 times and eventually got the Secret of 83 and the array location.







**Task 8: Make the Attack More Practical**

These tasks involve carrying out the full Meltdown Attack. The previous tasks have set us up and this file is able to compile successfully, and the secret value is displayed. An important note here is to add our found address into the main function of the [MeltdownAttack.c] file.

