Lab 2 Meltdown and Spectre Attacks

**In order to see all the attacks in this lab, we need to make sure the host machine runs on Intel CPUs. Students can team up with other students to do this lab. Please write the names of all the team members. Max group size is 4.**

Install Virtual Box and download the VM image of Seed Lab from Syracuse University (Select the Google Drive Link) – use the Ubuntu 32-bit 16.04 Version.

<https://seedsecuritylabs.org/lab_env.html>

Download all the code from the Blackboard into the VM. The sudo password for the VM is: dees

**Part I: Meltdown Attack**

**Question 1.** Compile the 1cachetime.c file using the following command and execute the program. Screenshot the results and attach to the report. Which two arrays have the lowest CPU cycles in access time?

Compilation command:

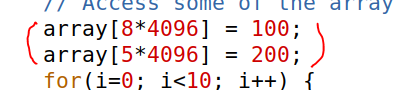
gcc -march=native 1cachetime.c

Text

Description automatically generated

Two arrays that have the lowest CPU cycles: [3\*4096] and [7\*4096]

b. Now, open the 1cachetime.c file and change the two arrays as shown in the following:



Re-compile and re-run the program. Which two arrays have the lowest CPU cycles in access time? Screenshot the results and attach to the report.

A close-up of a document

Description automatically generated with medium confidence

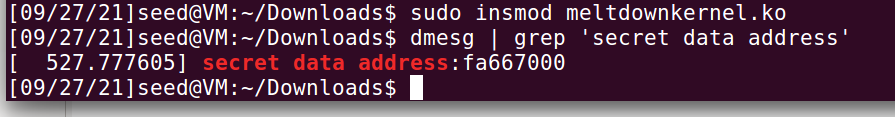
Two arrays that have the lowest CPU cycles: [8\*4096] and [5\*4096]

**Question 2**. Compile the 2flushreload.c using “gcc -march=native 2cachetime.c” and execute the program. By adjusting the CACHE\_HIT\_THRESHOLD, take a screenshot of the results with the secret and what is the CACHE\_HIT\_THRESHOLD value you have found on your computer.

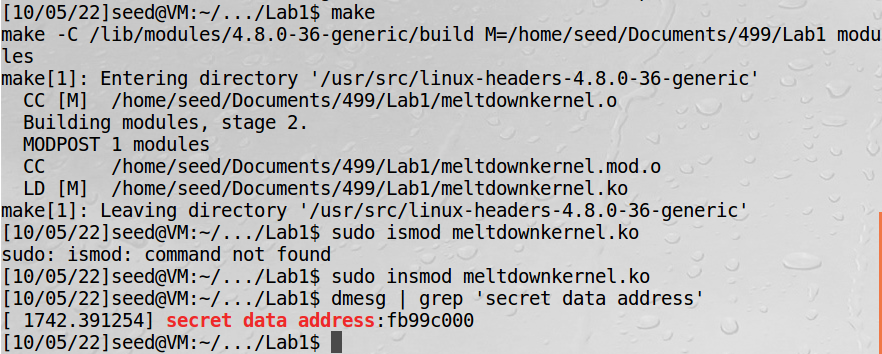
Graphical user interface, text

Description automatically generated

**Question 3.** Let us make the file first and then you will see meltdownkernel.ko is generated, Type “make” in the terminal. Make sure the make file is correctly done and then type in the following two commands:

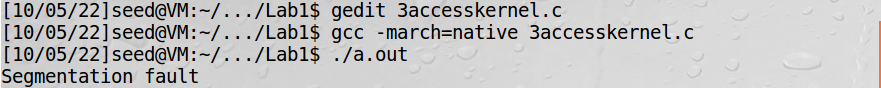


You will see an address. Here is mine: 0xfa66700. You should have a different one with mine. Open 3accesskernel.c and change the memory address with the one you found and compile the file with: “gcc -march=native 3accesskernel.c”. Screenshot the results.



Text

Description automatically generated



**Question 4.** Now open 4exceptionhandling.c and change the memory address with the one you found and compile the file with: “gcc -march=native 4exceptionhandling.c”. Screenshot the results and compare with the results from Question 3.

Graphical user interface

Description automatically generated with medium confidence

Text

Description automatically generated

In this screenshot, the program continues, but there is a memory access violation. In the last screenshot, the program failed to execute (hence segmentation fault)

**Question 5.** Now open 5meltdownexperiment.c and change the memory address with the one you found and compile the file with: “gcc -march=native 5meltdownexperiment”. Run several times and see how many times you can retrieve the secret number. Note that if none of them are successful, you might want to adjust CACHE\_HIT\_THRESHOLD. The correct result will be secret 7.

Text

Description automatically generated

**[Bonus + 1] Question 6.** Now in 5meltdownexperiment.c, let us change “meltdown(0x…….)” into “meltdown\_asm(0x…….)”, this will help us enlarge the attack time window by asking the CPU to do something. Re-compile the file, execute and screenshot the results. The correct result will be secret 83 S, which is the first letter of SEEDlabs.

Text, letter

Description automatically generated



**[Bonus + 1] Question 7.** Finally, let us compile the 6meltdownattack.c. Execute and screenshot the results. The correct result should be the entire word of SEEDlabs.

Text, letter

Description automatically generated

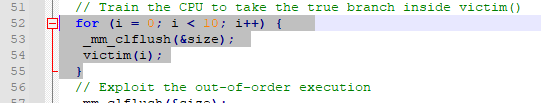
**Part II: Spectre Attack**

**Question 1**. Compile 8spectreexperiment.c with the “gcc -march=native 1spectreexperiment.c” and execute the program. Take a screenshot of the result. It should reveal the secret of 97.

Graphical user interface, text

Description automatically generated

b) Now recall that we have train the CPU using the loop from the number from 1-10. This makes the CPU to believe that x < size will be true so it will perform speculative execution.



Now, change victim(i) in Line 54 into victim(i+20). Re-compile and execute the program. Take a screenshot and explain. The attack should fail, and you should not see the secret of 97.

Text

Description automatically generated

**Question 2**. Compile 9spectreattack.c and execute the program. Take a screenshot of the result. It should reveal the Secret of 65 A. Then incrementally add displacement to “spectreAttack(larger\_x);”

larger\_x + 1, larger\_x + 2, …. and see whether you can recover all the characters in \*secret. Take screenshots of the results.

Text

Description automatically generated

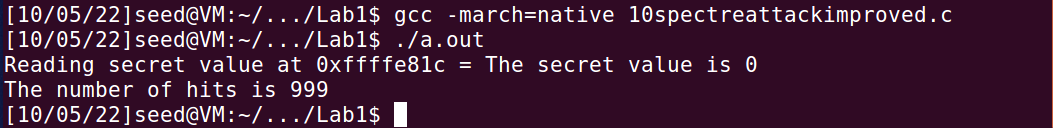
Text

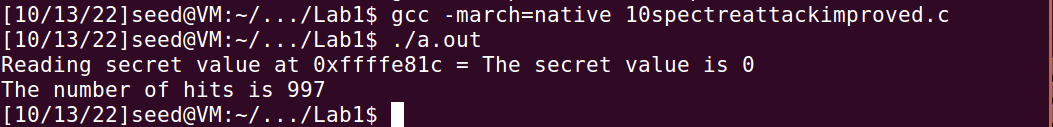
Description automatically generated

Above screenshot is after I increment by 4. Full message cannot display on my computer

**Question 3.** Compile 10spectreattackimproved.c and execute the program. Take a screenshot of the result and whether the attack is successful? If unsuccessful, you may want to adjust the value of CACHE\_HIT\_THRESHOLD. Then incrementally add displacement to “spectreAttack(larger\_x);”

larger\_x + 1, larger\_x + 2, …. and see whether you can recover all the characters in \*secret. Take screenshots of the results.





Above screenshot is after I increment by 3. Full message cannot display on my computer