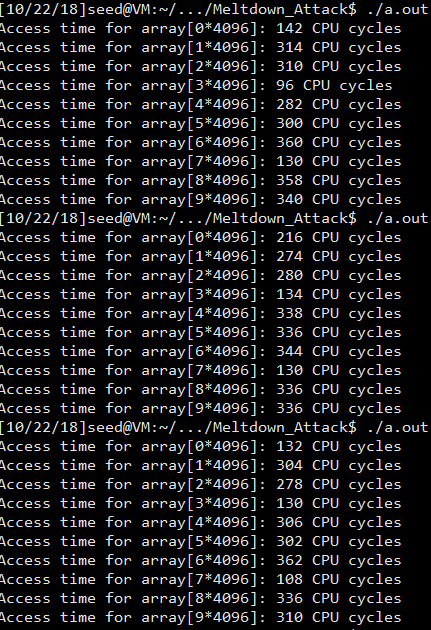
Meltdown Attack Lab

## Task 1: Reading from Cache versus from Memory

### Compile the server program and run

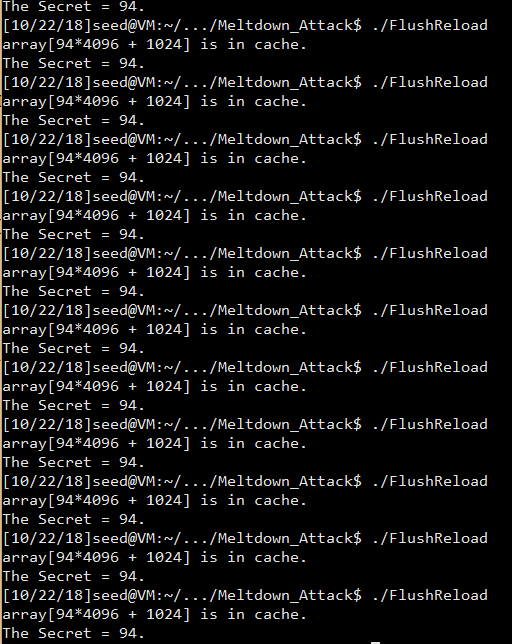
Three samples:



Observation and Explanation:

I find that [3] and [7] are not always faster than others, but they seem to have high possibility to run faster. Besides, [1] seems to performance great as well.

## Task 2: Using Cache as a Side Channel



Observation and Explanation:

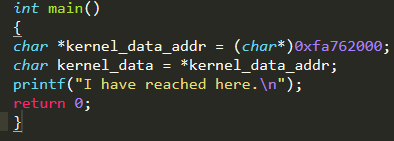
For all the results from 20 times, I always get the secret as 94. Though the secret number is only used in victim function, it will be read directly into cache because cache is always the faster choice. Thus, although we may not have information of function victim in main memory, we can use cache to test what indeed is used.

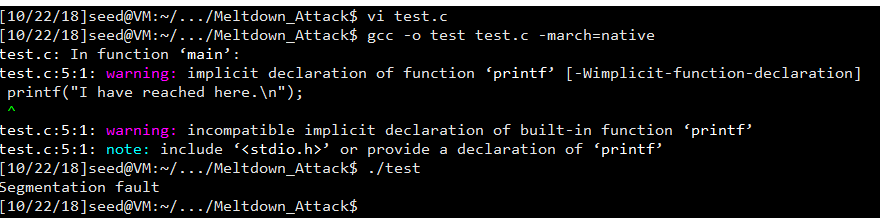
## Task 3: Place Secret Data in Kernel Space

After installing the mod, we can see that the address of the secret data is fa762000. We placed the secret data in the kernel space successfully.

## Task 4: Access Kernel Memory from User Space

My own code:



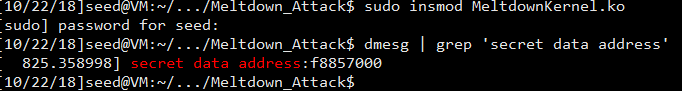


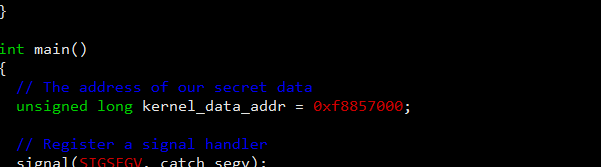
After my experiment, I find only the first line can be executed without segmentation fault. If I add the second line, the program will crash, because we access a kernel memory from the user space.

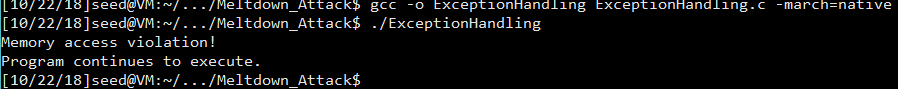
## Task 5: Change the Server Program’s Memory

Modify the code:

I restarted the machine before this task. So, now the new address is f8857000.



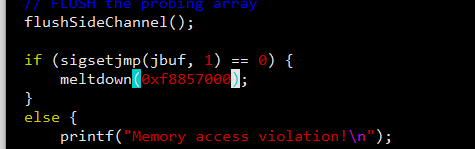


I find that the program is executed correctly. 

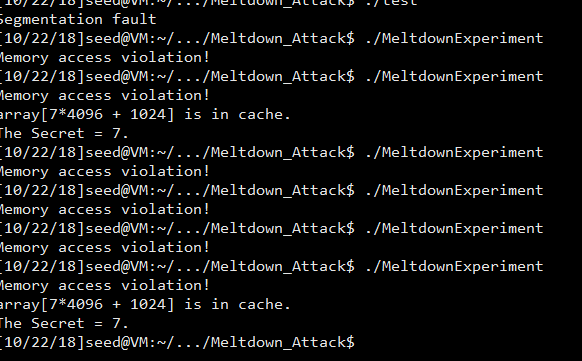
For each time execution, due to the handler, the program can jump back to the break point and execute from the else branch. Controlling the exception, we find the program can continue without crashed.

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

Modify the code:



After changing the address to 0xf8857000, compile and run



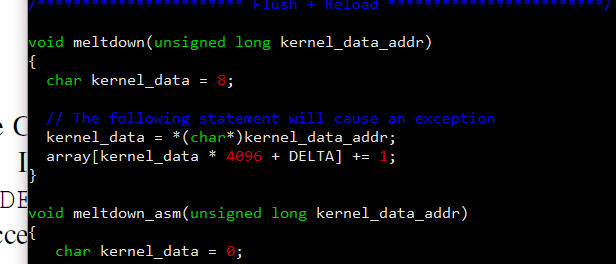
From the screen, I can see that the secret number is found in the cache, though it may be not found in most cases.

Seeing the result, we can see that the out of order is making effects. The biggest evidence that line 2 executes is that the secret we found is 7. The out of order execution exits in my CPU, but we do not have a hundred percent that the program is executed as we expected. That is why in most cases there is still memory access violation.

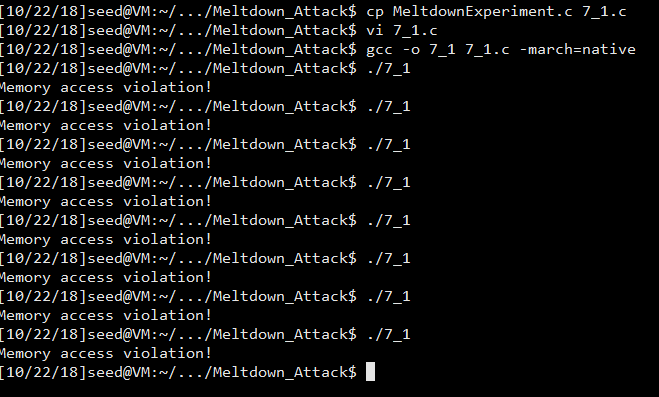
## Task 7: The Basic Meltdown Attack

### Task 7.1: A Naive Approach

Firstly, modify the code:



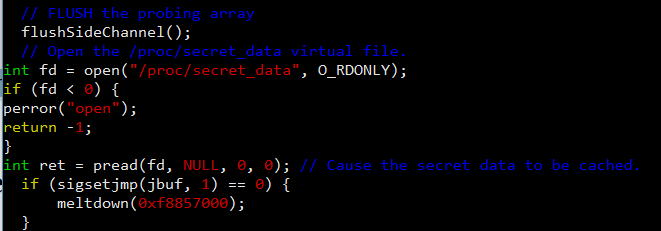
As above, I change the kernel\_data to 8, and change 7 into kernel\_data.



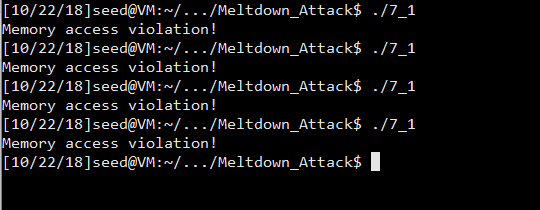
After several times trying, I find the program cannot find the secrete data in the cache directly. It seems that when the security check is done, the kernel data is still on its way from the memory to the register, the out-of-order execution will be immediately interrupted and discarded, because the access check fails. Our attack will fail as well.

### Task 7.2: Improve the Attack by Getting the Secret Data Cached

Modify:



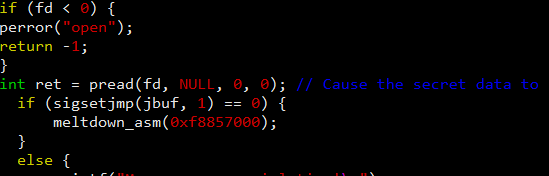
Test:



Still not succeed.

### Task 7.3: Using Assembly Code to Trigger Meltdown

Modify:

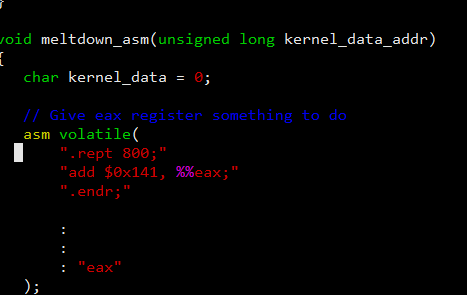


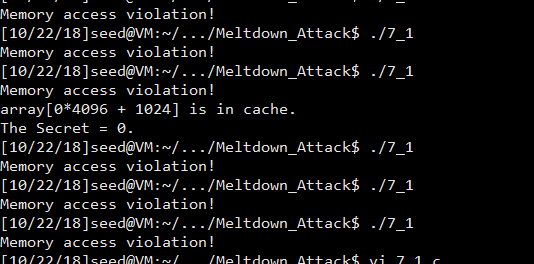
Test:

Only changing the code as above, the attack still cannot succeed.

However, if I change the number of loop, things become easy.

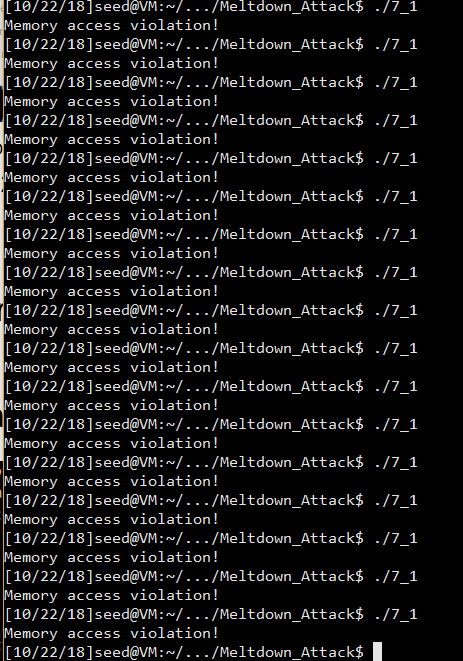
Change 400 to 800:





It seems that after 20 times of attempts, the secret is finally found.

Change the times of loop to 200:

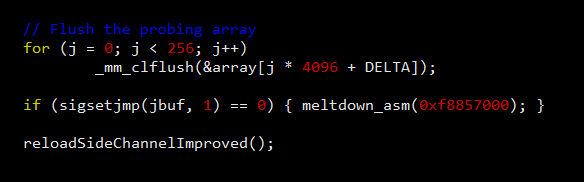


After over 30 times attempts, I am pretty sure that the possibility decreased.

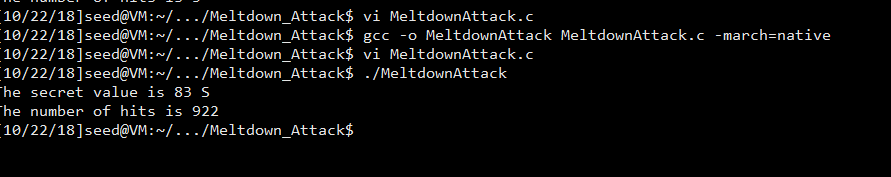
In the tasks, though there is small chance that the target is cached, after modifying the codes, we ensured that the target is cached and we win the race condition to steal the secret from the kernel.

## Task 8: Make the Attack More Practical

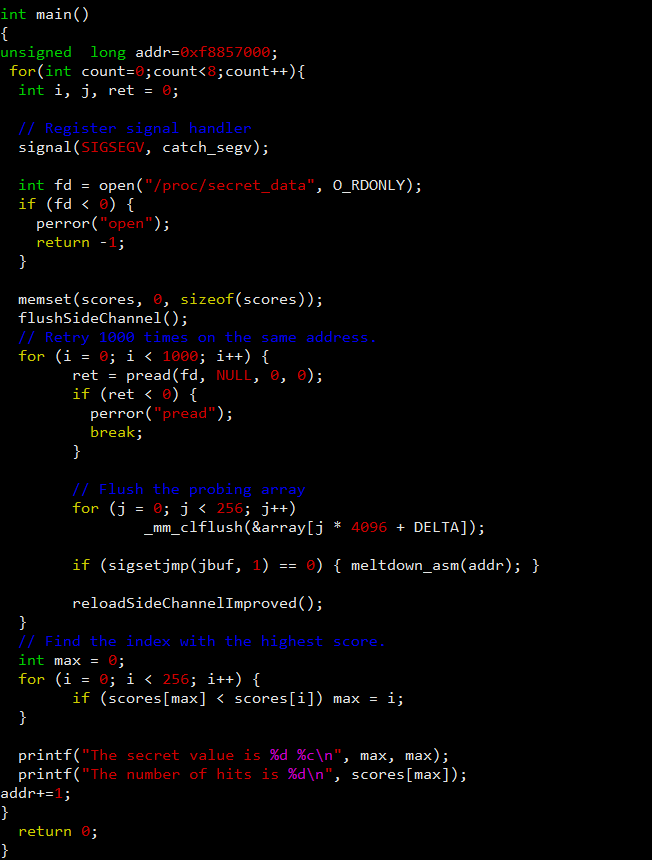
Modify the address:



Without changing other part of the codes, the result is as follows:

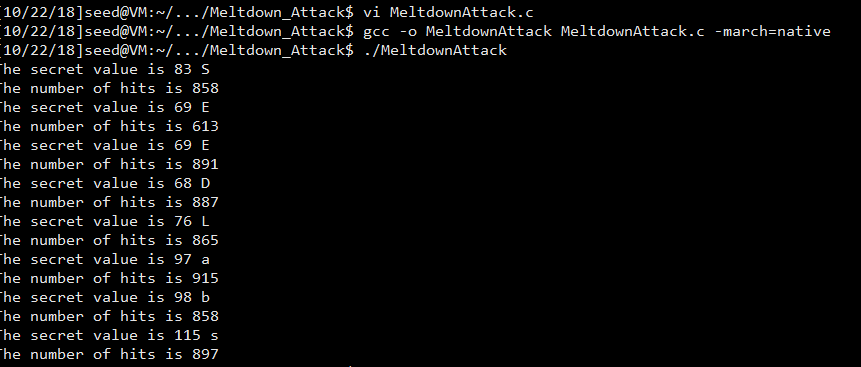


To print the full 8 bytes, we need to change a little bit of the codes:



It is easy. What we need to do is do the main function for 8 times and each time add one on the secrete address.

Result:



The secret is SEEDLabs.