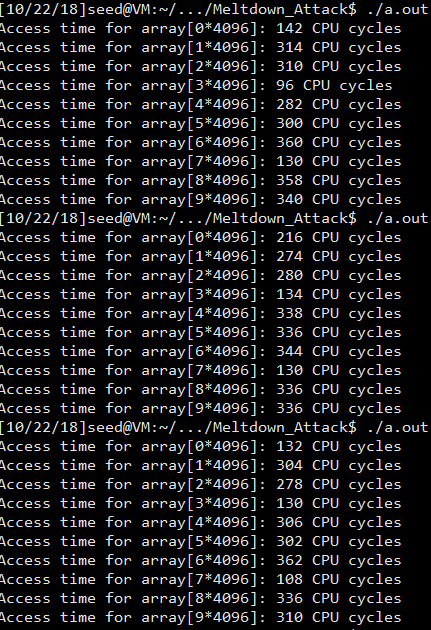
Spectre\_Attack Lab

## Task 1: Reading from Cache versus from Memory

### Compile the server program and run

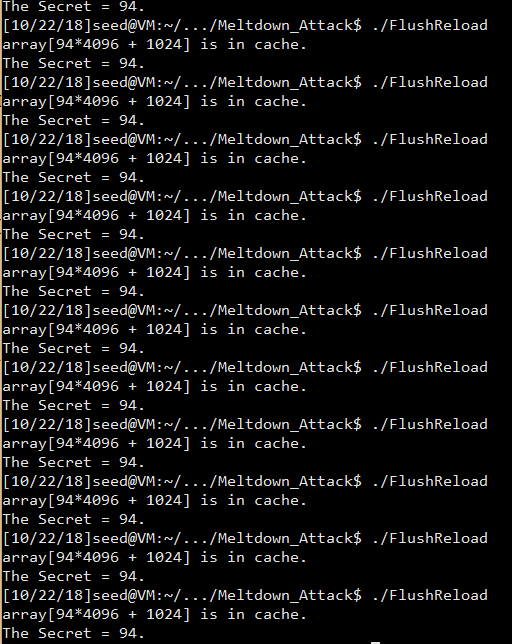
Three samples:



Observation and Explanation:

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

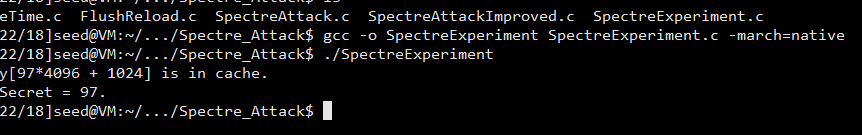
## Task 2: Using Cache as a Side Channel



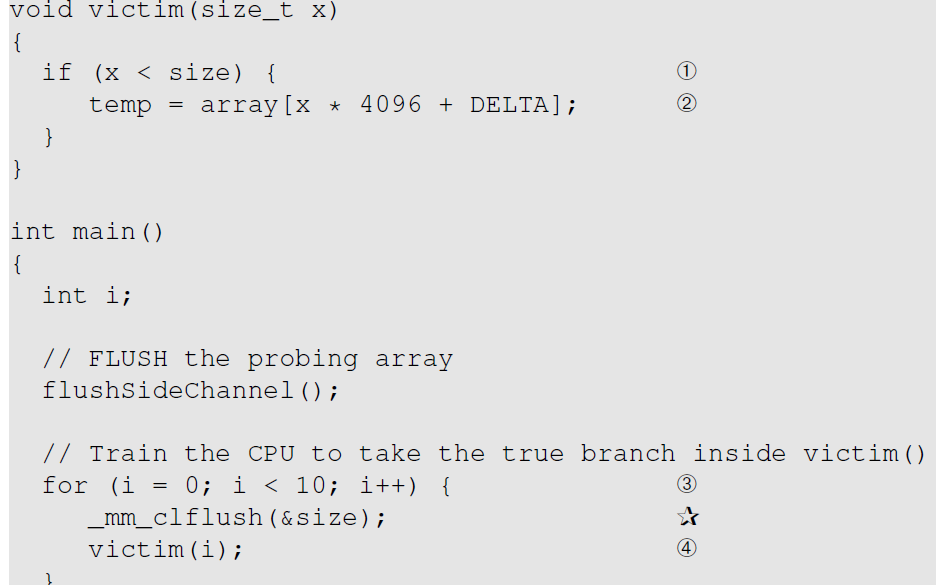
Observation and Explanation:

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: Out-of-Order Execution and Branch Prediction

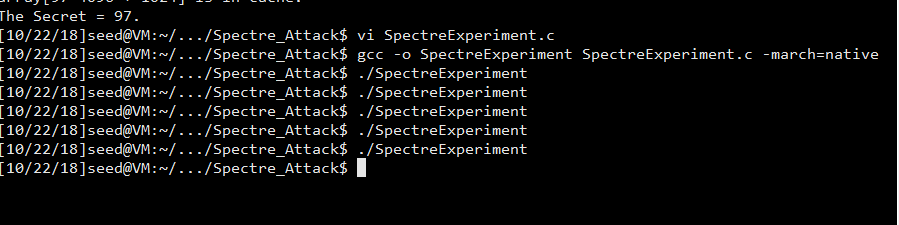


After Compiling and running the code, we find that the secret is shown. In the first part of the code, the CPU is trained.



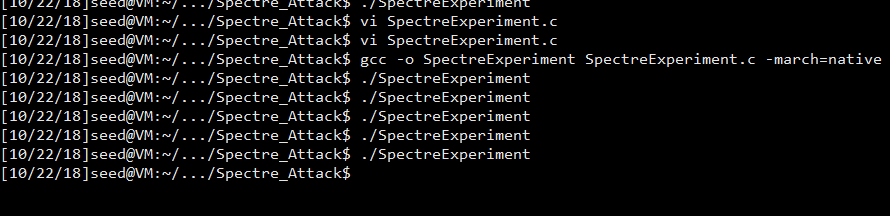
For each value, there is always x<size. Thus, the CPU is trained to guess true in the condition. And once the value is changed into secret 97, though the condition become false it can still execute the line 2 due to the out of order execution. Once the CPU find the value is false, it will discard the content in the if function’s field. However, there will be a copy in the cache. That is why we can succeed.

Comment the star lines:



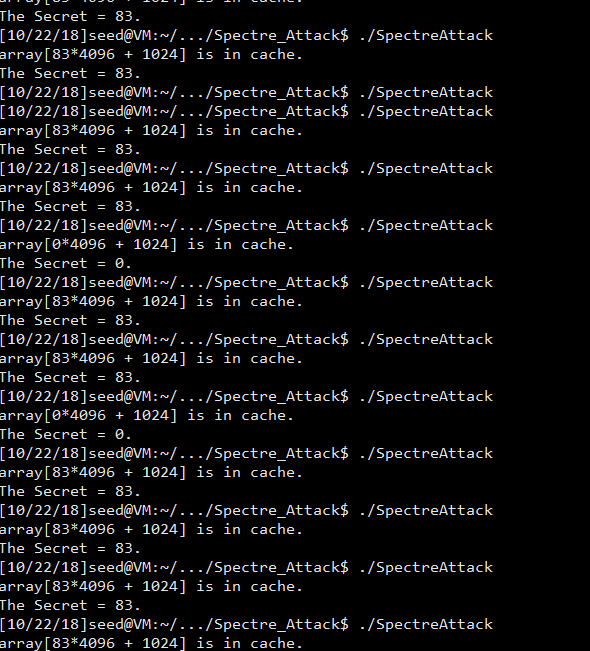
Obviously, after commenting the lines with star mark, we cannot get the secret. The function \_mm\_clflush cleans the size value in main memory. If we have the two lines, we cannot find size in the main memory. Thus, each time we need to read the size value from the stack, which takes some time. With the time, out-of-order execution occurs to execute the program faster, so the CPU trains. Out-of-order execution only happens when the current instruction stores and if we have the value in the maim memory the execution will be fast and not stunk in finding size.

Replace Line 4 with victim(i + 20);



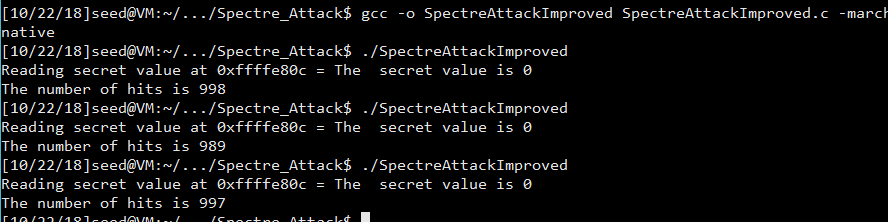
As well, we don’t find the secret. Changing to i+20, the condition will always be recognized as false. So, the train result of CPU is fault, which means the out-of-order execution will not execute the program inside the if field. Instead, it may execute the return function because CPU think the function has big possibility to return.

## Task 4: The Spectre Attack

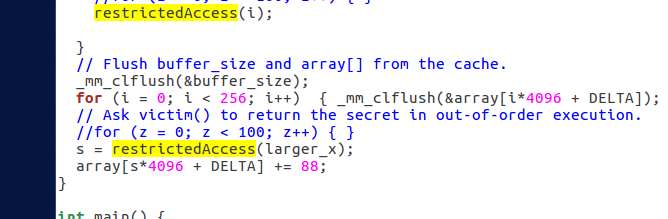


We can see that although the result always be 83, which is the character ‘S’, there is some case that the secrete we find is 0. From the function reload, we can see that the print function makes effects when time2 <= CACHE\_HIT\_THRESHOLD. There could be some noise that in the cache as 0.

## Task 5: Improve the Attack Accuracy

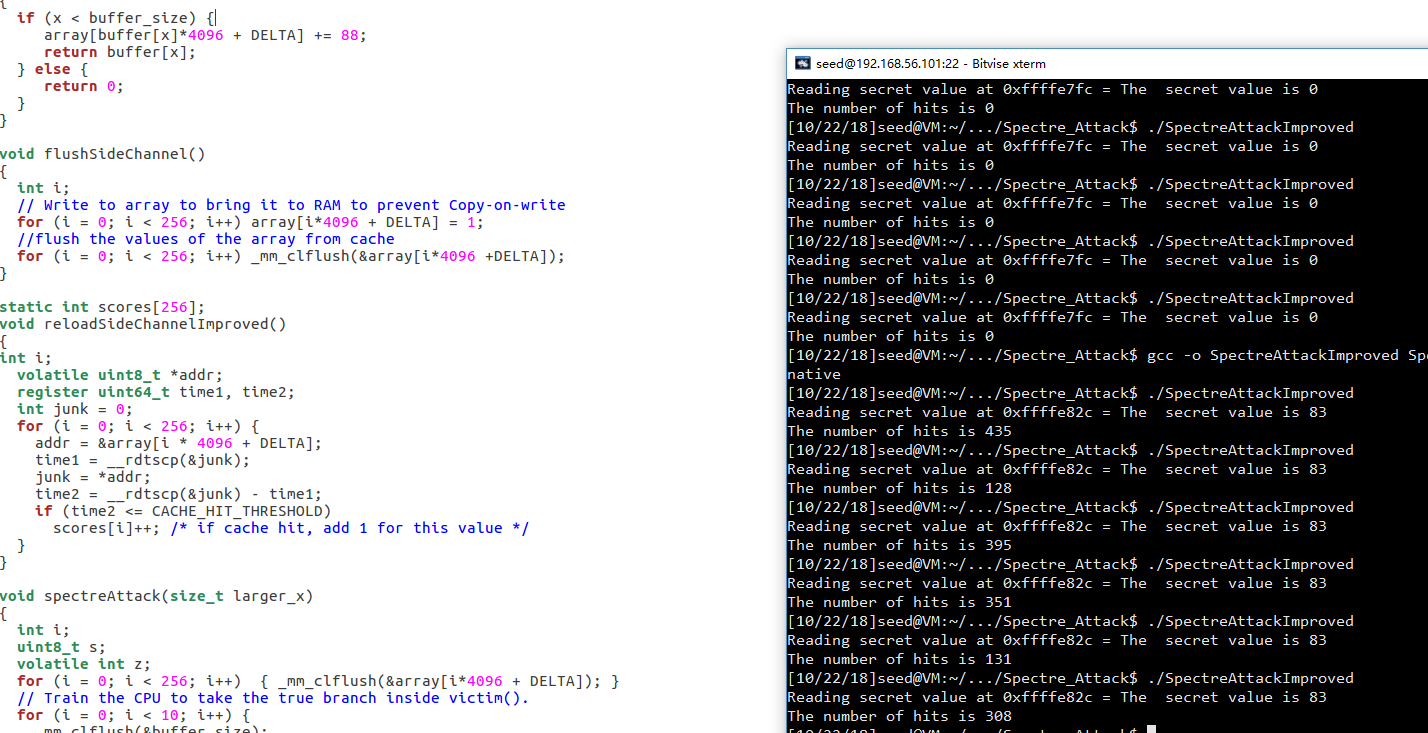


Running the code given, I find that each time I get secret value as 0. Thus, I need to fix the program.



The wrong is here. Although there may be out-of-order execution, the return value cannot change. The only change we made is in the cache. If the if condition is calculated as false, the system will discard the value and return 0 as well.

Remove the parameter s and move the last line into the restrictedAccess function:



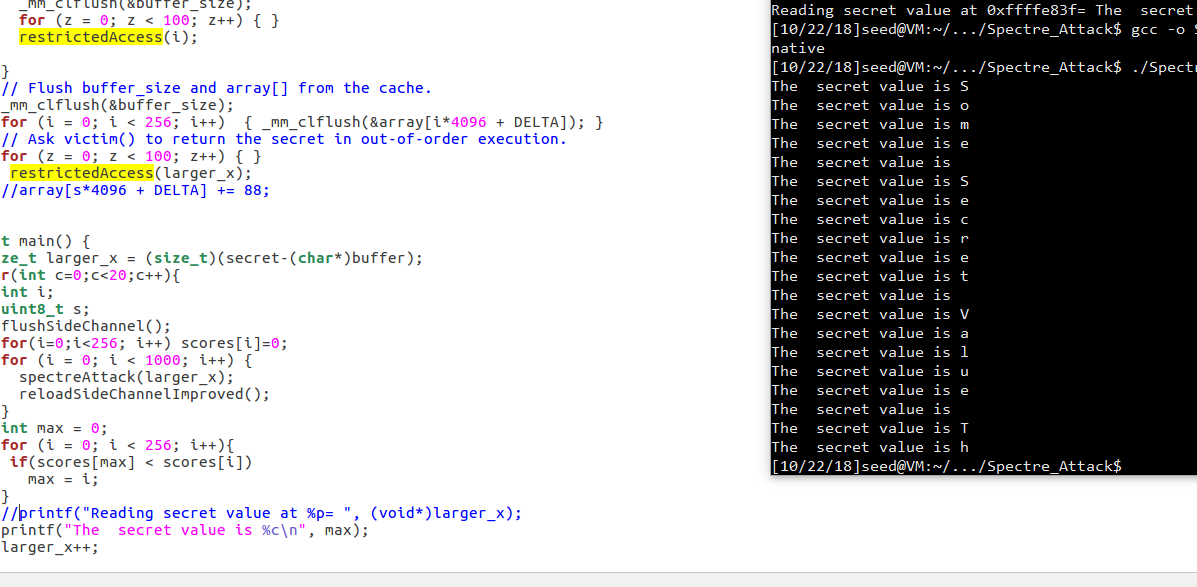
As we can see, after modifying the codes, each time the secret value will be shown correctly.

## Task 6: Steal the Entire Secret String

In this section, what we need to do is to print out the full string.

Using a for loop, we can simply print out all the characters.

Each char is one byte, so the pointer of larger\_x should +1 for each character.



As we can see, after adding the for loop, we can see the values: Some Secret Value.