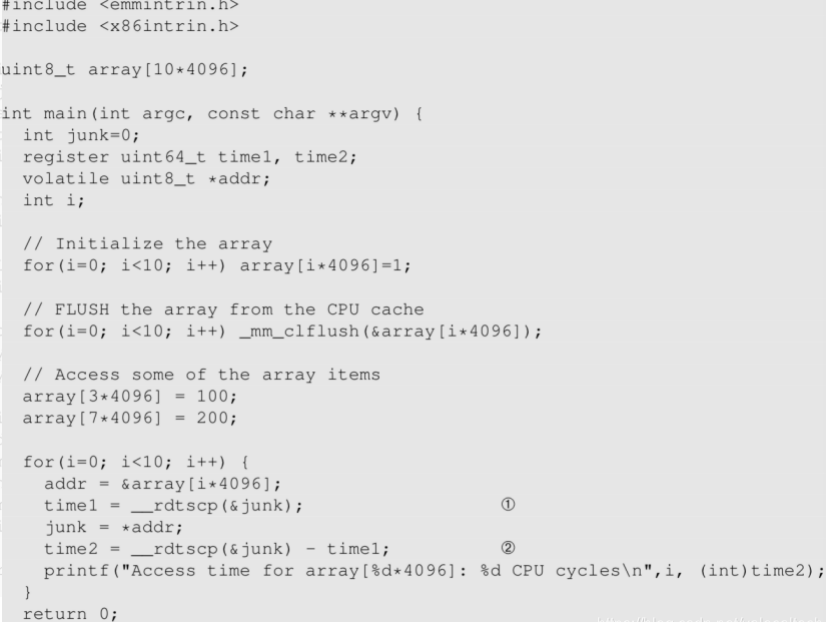
(1)在代码中，我们有一个规模为104096的数组。我们首先访问其中两个元素，分别是array[3\*4096]和array[7\*4096]。因此，包含着两个元素的页就会被cache

然后我们读取元素，从array[04096]到array[94096]并测量读取所花费的时间

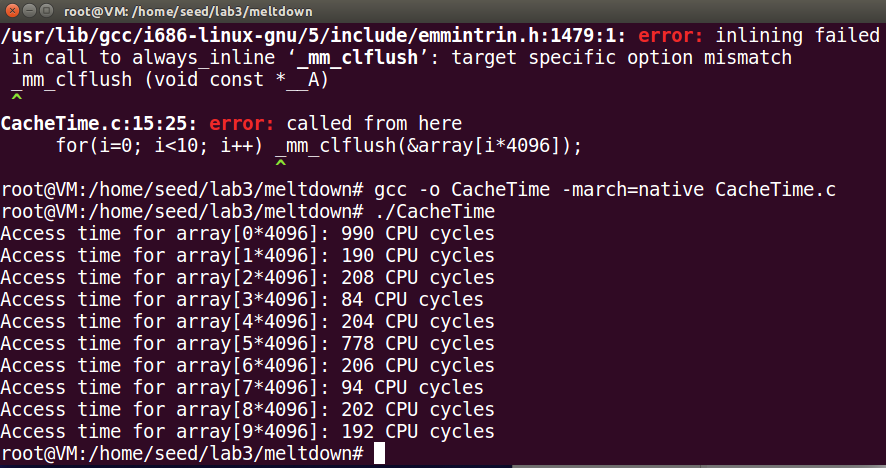
在代码中，（1）在内存读之前读取CPU的时间戳（TSC），

而（2）在内存读之后读取计时器。他们的差值就是花在内存读上的时间。

需要注意，cache在是cache block层次上完成的，而不是byte层次。一个典型的cache block是64字节。我们使用array[k\*4096]，所以程序用到的元素不会有两个落在相同的cache block中



寻找 cache 时间阈值，编译并运行 CachTime.c，运行程序 10 次，观察输出，通过 实验可以设定阈值为 150。



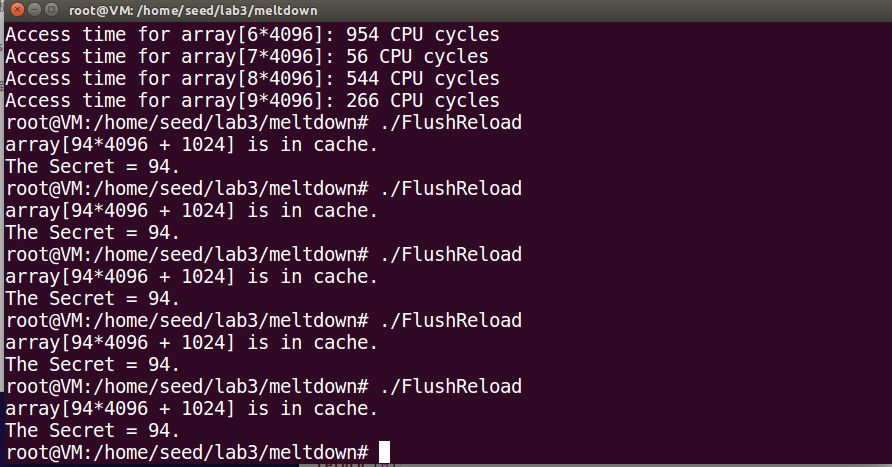
可以看到读取array[3\*4096]和array[7\*4096]时要比读取其他元素快得多

我们来尝试通过侧信道提取victim function会使用的secret value

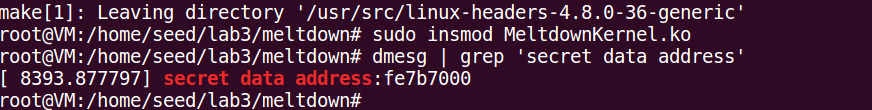
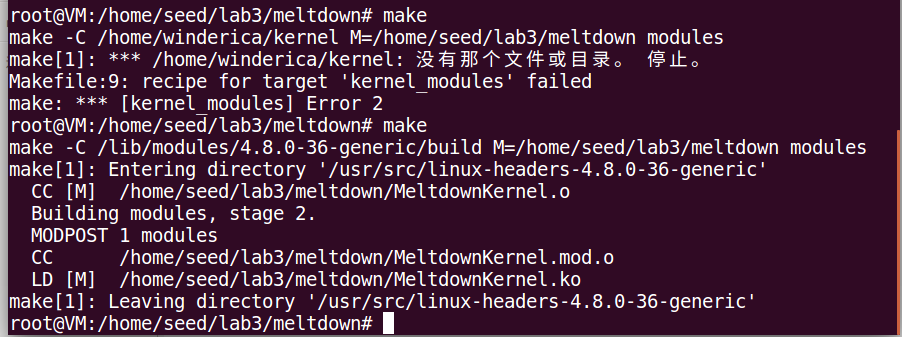
假设victim function使用secret value作为索引加载array中的某个元素，同时假设该sexret value不能从外部访问到。我们的目标是使用侧信道获得这个secret value

我们使用的技术为FLUSH+RELOAD

编译并运行 FlushReload.c，运行程序多次，得到秘密值 94。

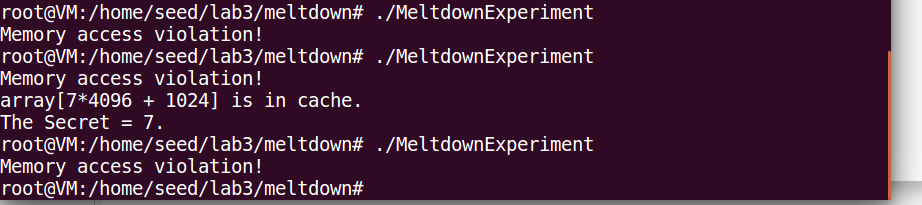


编译 MeltdownKernel.c，按照该内核模块，并使用 dmesg 发现秘密数据的地址。

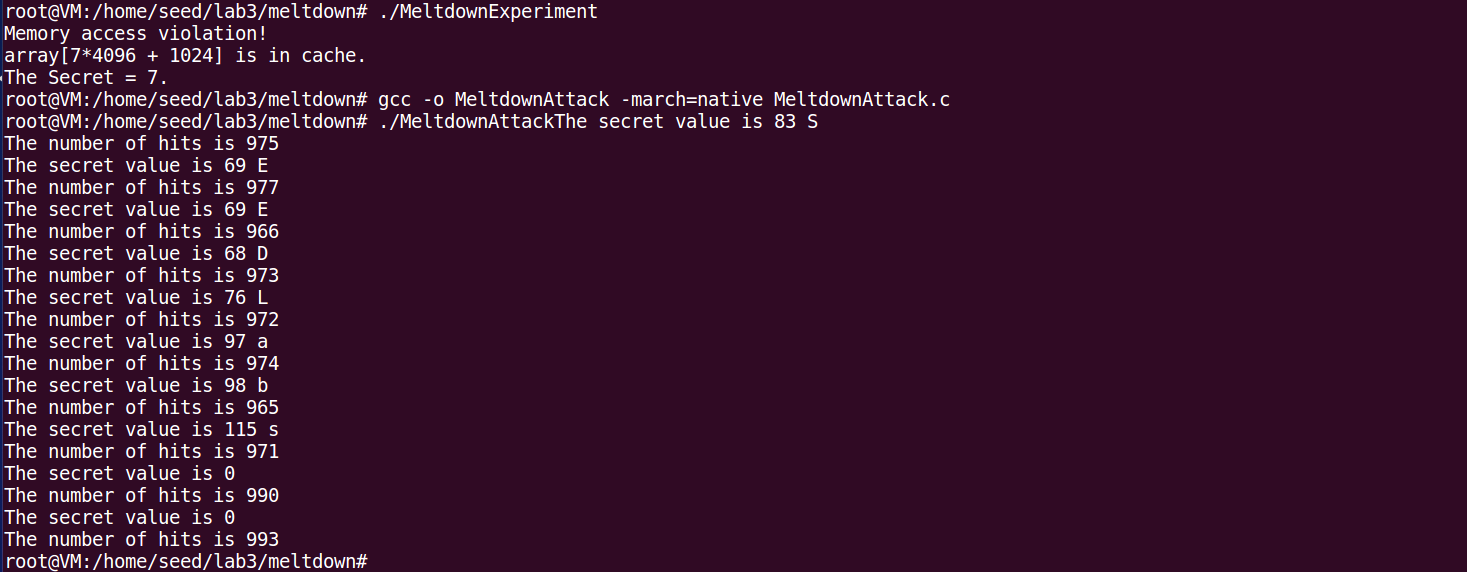


编译并运行 MeltdownExperiment.c，可以获得秘密值。

然后编译执行 MeltdownExperiment.c，找到 cache 中的数据，代表 array[7\*4096 + 1024]是在 cache 中，被高速读入了内存



编译并执行 MeltdownAttack.c，记录并解释你的观察。该代码只窃取内核的一个字节的秘密。上述内核模块中的实际秘密有 8 个字节。请尝试修改上面的代码以获取所 有 8 个字节的秘密数据。

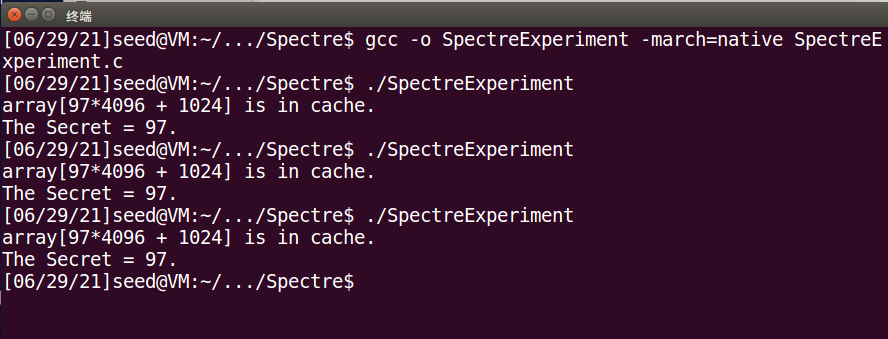


编译并运行 SpectreExperiment.c，运行程序多次。

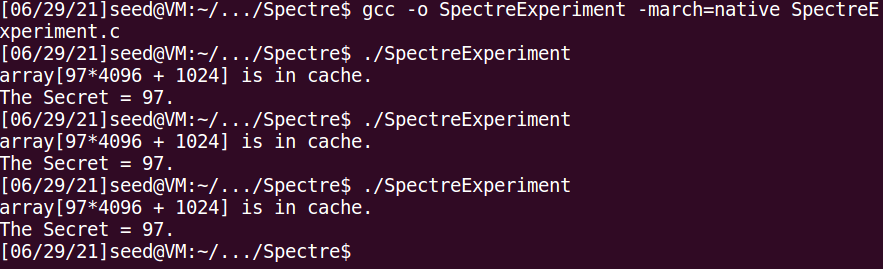
注释掉★部分后，程序没有输出。

改为 i+2 后，程序没有输出

可以看出，原本的程序攻击成功率很高，而注释之后成功率很低。注释掉的星号行的功能都是清空缓存中的 size，延长获取 size 的时间，从而延迟 discard 的发生，给窃取数据留下足够的窗口期。因此注释之后，攻击的成功率大大降低了。 编译并运行 SpectreAttack.c，运行程序多次，观察是否有一致性的输出



编译并运行 SpectreAttack.c，运行程序多次，观察是否有一致性的输出



有的时候是 0，有的时候是 83。83 是大写字母 S。对比 SpectreAttack.c 和 SpectreExperiment.c 可以发现，在 SpectreAttack.c 中如果是 false 分支会返回 0（随 后 0 将被访问、载入缓存，从而 reloadSideChannel 返回 0。换言之，屏幕显示 Secret 为 0 并不代表秘密值真的为 0，而是表示攻击失败）。在 SpectreExperiment.c 中如果是 false，则不会有数据被载入缓存。因此该结果的解释是，有的时候走了 false 分支， 有的时候走了 true 分支，即有的时候攻击成功，有的时候攻击失败。

