**Spectre漏洞测试报告**

1. **漏洞分析**

Spectre漏洞主要利用了转移猜测的不确定性以及从cache中读取数据与从内存中读取数据的时间差来进行数据分析。

CPU在执行分支语句时会通过预测的方法估计这条指令可能的结果，并且采用预先计算的方法来加快处理过程。如对语句

if (x < array1\_size)

y = array2[array1[x] \* 512]

进行分析时，首先要判断x与array1\_size的大小，但是对array1\_size的数值进行读取会耗费大量的时间，于是CPU就会根据之前的经验对这条语句的结果进行猜测。当攻击者有意识的通过大量的例子迷惑CPU后，在执行攻击代码时CPU就会误以为本次判断也是通过，于是将执行后续的操作。如果最终发现x大于array1\_size，这条语句将会被终止，并且此时才会将后续的y的赋值进行撤销。

虽然后续执行中对array1[x]的读取会因为x的越界而被标记为异常，但是系统还是会读取到array1[x]的值，并且通过这个值去进行后续的操作。虽然异常在最后ROB提交阶段会被处理掉，但是由于已经读取到了array1[x]的值，并且通过对这个值乘上页面大小进行处理，就可以对array2数组进行操作。

假设开始时array2数组不在缓存中，当读取到array2[array1[x]\*512]时，相应位置的页面就会被加载到缓存中去。直接从cache中读取和从内存中读取有一定的时间差，通过时间差这个侧信道信息对array2数组进行遍历判断就能得出是哪个位置被加载到了缓存中，从而也就得到了array1[x]的值，从而得到秘密信息。

这个漏洞利用了异常指令撤销后残存在cache中的信息以及读取的时间差。由于正常的访存过程中也会存在分支指令转移猜测失败的情况，所以全面禁止转移猜测的优化过程就会影响系统的性能。

1. **测试平台**

机器1:

CPU型号 Intel i5;

操作系统 Windows;

编译工具 Visual Studio 2017

机器2:

CPU型号：Intel(R) Core(TM) i5-4210U CPU @1.70GHZ 2.40GHZ

操作系统：Ubuntu 14.04.1\_LTS X86\_64

编译工具：GUN GCC compiler

机器3:

CPU型号 Intel i7-7700HQ;

操作系统 Windows 10;

编译工具 Visual Studio 2017

机器4:

CPU型号 Intel i3 3217U;

操作系统 Windows 10;

编译工具 Visual Studio 2017

1. **代码分析**

**3.1重要变量：**

|  |  |
| --- | --- |
| 变量名 | 意义 |
| secret  array1[] | 表示我们要窃取的密码（地址）  普通数组（地址） |
| array2[] | 普通数组（地址） |
| malicious\_x | Secret与 array1两指针之间的差值，我们可以通过array1[malicious\_x]来读取secret |
| value[2] | value[0]表示侦测出来最高分对应的字符，value[1]表示第二高分对应的字符 |
| results[] | 标记记录命中次数 |
| score[2] | score[0]表示最高分，score[1]表示第二高分 |
| tries | 尝试次数 |

**3.2主函数main:**

**Step1**:计算机密码数组secret指针和普通数组array1指针的差值malicious\_x（在后面就通过差值进行越界访问）

**Step2**:定义value数组存储越界访问得到的值，定义score数组存储越界访问分数

**Step3**:初始化array2，在内存中为其分配地址空间

**Step4**;检测是否存在自定义输入

**Step5**:进入while循环利用漏洞进行攻击：

(1)调用readMemoryByte函数利用漏洞获取secret的值

(2)输出获取结果

**3.3函数readMemoryByte：**

(参数：malicious\_x，value，score)

作用:对系统进行训练以及攻击处理

**Step1**:定义results数组记录array2在cache中的命中情况，定义时间变量记录对array2的访存时间

**Step2**:初始化results

**Step3**:进行999次训练和攻击：

1. 首先清除掉array2数组以及array1数组的在cache内的记录。然后函数对CPU的分支预测进行了训练。30个循环，每训练5次，执行攻击1次。

(2)for循环进行训练和攻击：

当j为6的整数倍时，x为全1， x为malicious\_x，

当j不是6的整数倍时，x为全0, x为raining\_x，

调用victim\_function函数，进行训练或者攻击

详细推导过程如图3-1所示

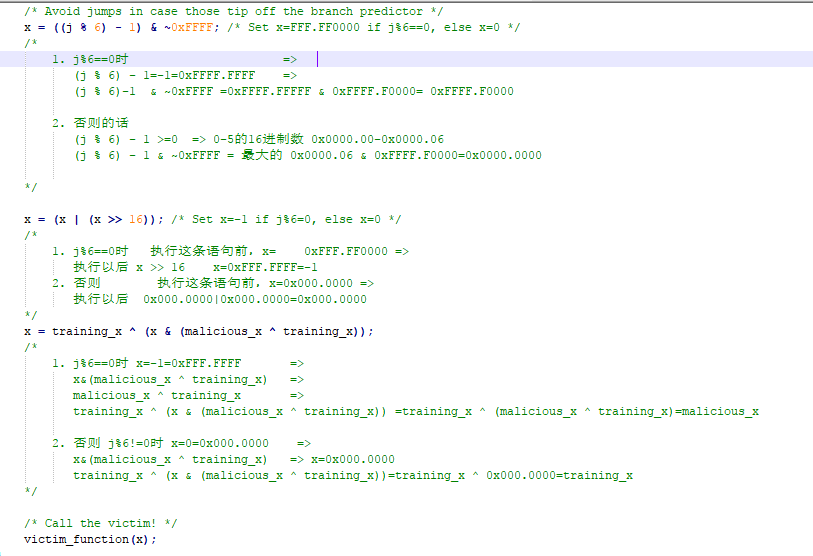


图3-1

**Step4**: 这部分代码，是为了不让 i 以1,2,3,4,5,6....的正常顺序做循环，避免stride prediction ,使用 ((i \* 167) + 13) & 255进行映射，可以让得到的所有i值是乱序且不重复的，可执行图3-2中的程序进行体验。

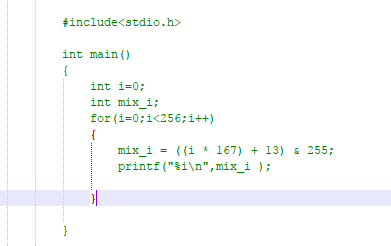
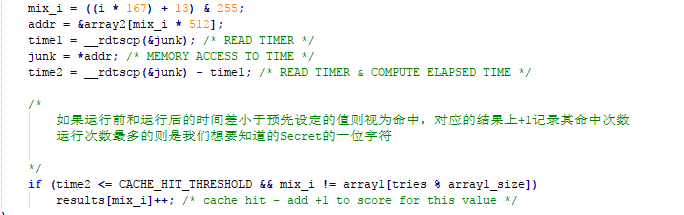


图3-2

for循环遍历array2，通过比较读取array2数组的时间来判断哪个值在缓存内，即命中了cache，在results对应的索引处加1，表示命中次数增加一次。如图3-3。



如图3-3

**Step5**：for循环根据result的记录值获取攻击结果

(1)找出result中最大元素的，索引值为secret的值，元素值为命中分数

(2)将值分别赋给value和store

**3.4函数victim\_function**：

该函数将用于越界读取分析。若不存在转移猜测，根据定义程序只能读取array1[0]~array1[15]的内容。但是实际运行时存在转移猜测的优化情况，于是程序就能读取整个array1数组。

**Step1**判断x的值是否小于array1\_size

**Step2**如果小于则取值 array2[array1[x] \* 512]

漏洞在于：当转移预测出错时，CPU已经将array2[malicious\_x \* 512]加载到cahce。

1. **测试结果**

不同的CPU，以及同一个CPU，在不同的时间内，得到的结果会稍有不同，下图是在intel i5下运行得到的结果。如图4-1存在second best的情况而图4-2都是best结果。

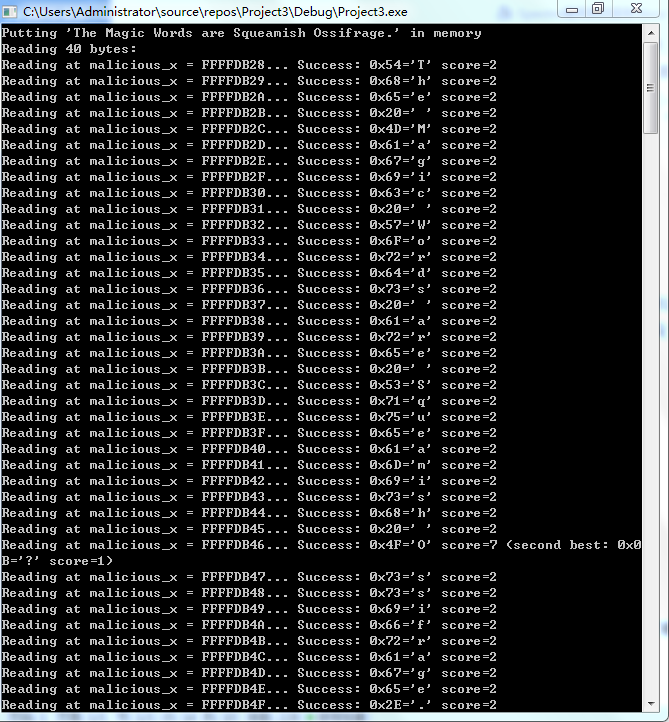


图4-1

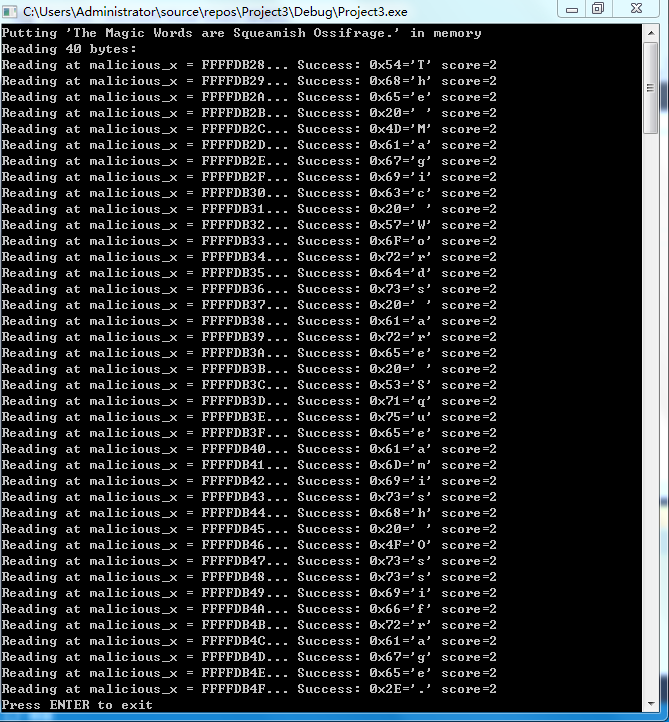


图4-2

在另一台PC（Intel(R) Core(TM) i5-4210U CPU @1.70GHZ 2.40GHZ）那个执行的结果，如图4-3所示，结果类似。

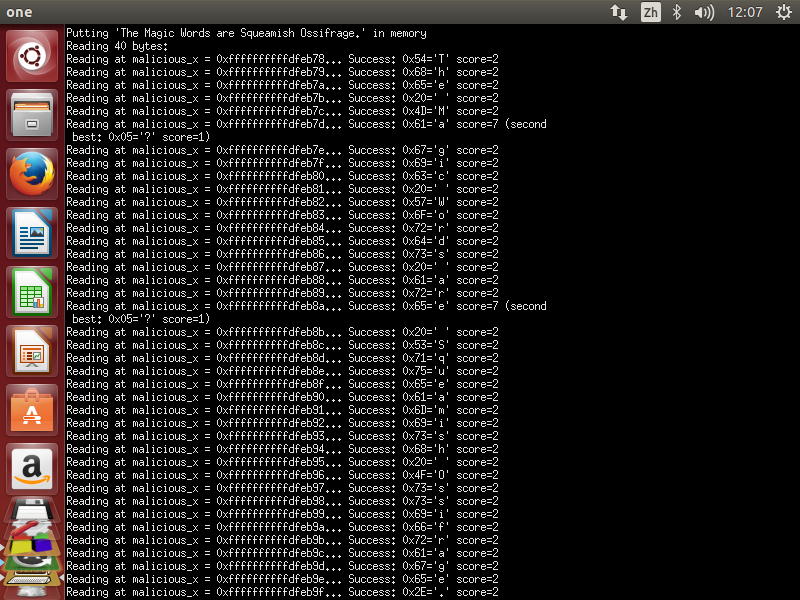


图4-3

1. **修复方法**
2. 操作系统厂商已经发布了补丁更新，但是有文章表示修复补丁会影响CPU的性能。

2.360 已经给出相关的修复程序

1. **待解决问题**

mm\_clflush()函数会存在跨平台不适用的情况。

1. **参考文献**

主要代码的来源链接：

<https://github.com/Eugnis/spectre-attack>

主要原理解释文章：

<http://bbs.nga.cn/read.php?tid=13191205&_fu=1779456%2C3&from=groupmessage&isappinstalled=0&rand=358>

<https://mp.weixin.qq.com/s/KmpN_oLNouXraoNd7kUgAw>

<https://www.zhihu.com/question/265012502/answer/290018306?utm_source=wechat_session&utm_medium=social>