Spectre攻击验证 实验报告

计97 朱美霖 2019013294

实验目的

分支预测是一种`CPU`优化技术。当分支指令发出之后，无相关优化技术的处理器，在未收到正确的反馈信息之前，不会做任何处理；而具有优化技术能力的`CPU`会在分支指令执行结束之前猜测指令结果，并提前运行预测的分支代码，以提高处理器指令流水线的性能。如果预测正确则提高`CPU`运行速度，如果预测失败`CPU`则丢弃计算结果，并将寄存器恢复至之前的状态。但这种性能优化技术是存在安全漏洞的，在预测失败的情况下`CPU`是不会恢复缓存状态的，因此可以利用分支预测技术读取敏感信息，并通过缓存侧信道泄露出来。

通过实现`Spectre`攻击样例，我们能学习了解在`CPU`中性能优化技术所带来的安全问题，进一步理解`CPU`的工作流程，加深对处理器硬件安全的认识。

实验内容

环境设置

- 操作系统：`macOS Monterey Version 12.4`
- 处理器型号：`Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz`

实现步骤

实现步骤与教材中的步骤相同，这里不赘述；具体的代码实现参考了`Spectre-attack`论文的源码，标注一下需要注意的点：

- 步骤（4）中用正确的索引调用`victim_function`5次，再用`malicious_x`调用`victim_function`一次的过程中，需要注意不能想当然地使用`if/else`判断循环次数而跳转操作，而需要巧妙地利用位操作完成判断`j%6`的过程：

  ```
  x = ((j % 6) - 1) & ~0xFFFF;
  /// if j % 6 == 0: x = 0xffff0000
  /// else x = 0x00000000;
  x = (x | (x >> 16));
  /// if j % 6 = 0: x = 0xffffffff;
  /// else x = 0x00000000;
  x = real_x ^ (x & (malicious_x ^ real_x));
  /// if j % 6 = 0: x = real_x ^ (malicious_x ^ real_x) = malicious_x;
  /// else x = real_x ^ 0x00000000 = real_x;
  ```

  这个奇怪的数学过程源自论文源码；

- 步骤（5）乱序读取`cache_set[]`的方法是一种极其简易的线性同余随机数生成器：

  ```
  for (i = 0; i < 256; i++) {
              random_i = ((i * 167) + 13) & 255;
  ```

其中`167`为质数，导致对于`0`到`256`的计算结果同样为`0`到`256`且不重复。这个trick同样来自论文源码；

- 对我困扰最大的问题是需要考虑`main`函数的`cache line padding`问题。在修改论文源码时，我发现如下的问题：`uint8_t value[2]`和`int score[2]`本身并不需要传参至`main`函数中打印，结果可以直接在每次的`attack`过程后直接`print`到控制台；但我在删除`main`函数中声明的这两个变量后，导致结果中的部分字符掉到了分支预测命中率的第二，而原代码跑下来的结果是所有字符都在命中率的第一；在与同学交流后，我们认为这可能由于源代码中在`main`函数内声明的变量组合在一起正好填满了`cache line`的大小（64B）而导致的。如果改变声明的变量后，可能造成处理这些变量时`fetch`到的`cache line`中有部分`cache_set`中的元素，从而影响分支预测的命中率。在经过消融实验后，我对`cache line padding`部分做了如下修改：

```
typedef struct MainVar {
    size_t malicious_x;
    size_t i;
    int64_t len;
    int64_t pad[5];
} __attribute__((aligned(4096))) MainVar;

int main() {
    MainVar x;
    ...
}
```

其中`__attribute__((aligned(4096)))`保证了该数据结构对于`cache`保持了`4KB`的`padding`，这个数被我设置的尽量大，来保证避免`Intel`固有的`DCU Streamer Prefetcher`机制会`prefetch`下一个`L1 cache`的`cache line`以导致的`cache line`不对齐的问题。在消融实验中，`aligned(128)`已经可以保证结果的正确性。

实验结果

```
❯ gcc spectre.c -o sp && ./sp
Secret Cybersecurity Fundamental Tsinghua University at: 0x102ffdf60
C   996     989
y   998     987
b   997     986
e   996     980
r   997     981
s   996     952
e   997     969
c   997     987
u   997     974
r   995     971
i   996     989
t   999     991
y   989     947
    998     979
F   998     966
u   998     976
n   994     978
d   995     969
a   996     977
m   997     983
e   994     937
n   994     968
t   998     989
a   992     980
l   996     967
    995     955
T   995     985
s   998     942
i   997     984
n   993     969
g   997     984
h   998     983
u   996     985
a   992     966
    996     993
U   998     970
n   995     966
i   996     981
v   997     992
e   994     970
r   999     943
s   998     969
i   995     981
t   998     978
y   992     954
```

可以看到利用`Spectre`漏洞准确窃取了我们存储在`char* secret`中的密文内容`Cybersecurity Fundamental Tsinghua University`。