spectre-v4笔记

- 资料
  - AMD前瞻执行之内存访问预测器逆向分析与漏洞利用 (qq.com)
  - RISC-V CPU侧信道攻击原理与实践（7）---Spectre-V4 - 知乎 (zhihu.com)
- 内存消歧（Memory Disambiguation)技术：高性能乱序处理器采用该技术来高效地执行存储器相关的load和store操作，处理器通过一组内置逻辑电路来检测这些存储操作的真、假依赖关系，通过消除假的依赖关系来充分利用CPU的指令并行性，假如发生依赖关系判断错误则需要能够从错误中恢复过来。当处理器试图乱序执行指令时，处理器必须尊重指令之间的真正依赖关系。
-

```
// overwrite data via different pointer
// pointer chasing makes this extremely slow
(*(*data_slowslowptr))[x] = OVERWRITE;

// data[x] should now be "#"
// uncomment next line to break attack 取消注释下一行可以打断攻击
/*当取消注释后，mfence() 会确保数据覆盖操作 ((*(*data_slowslowptr))[x] = OVERWRITE;) 完成后
所有后续的读取操作不会使用旧值（即原始的 SECRET 值），从而避免了信息泄露。*/
// mfence();
// Encode stale value in the cache
cache_encode(data[x]);
/* defenition for cache_encode() 可以访问特定内存位置
void cache_encode(char data) {
maccess(mem + data * pagesize);
}
*/
```

- mem = (char *)(((size_t)_mem & ~(pagesize - 1)) + pagesize * 2);// page aligned 页面对齐，并偏移两个页面

1. `(size_t)_mem`：
   - 将指针 `_mem` 转换为 `size_t` 类型，以便进行位运算。`size_t` 是无符号整型，通常用于表示内存大小或指针地址。
2. `pagesize - 1`：
   - 计算页面大小减去1。假设 `pagesize` 是4096字节，那么 `pagesize - 1` 将是4095（即 `0xFFF`）。
3. `~(pagesize - 1)`：
   - 对 `pagesize - 1` 取反，结果是一个掩码，用于清除指针地址的低位部分，使得指针地址向下对齐到最近的页面边界。
   - 例如，对于4096字节页面，`~(pagesize - 1)` 将是 `0xFFFFF000`（假设是32位地址空间）。
4. `(size_t)_mem & ~(pagesize - 1)`：
   - 将 `_mem` 地址与掩码进行按位与操作，清除地址的低位部分，确保结果是页面大小的整数倍。
   - 这将确保 `mem` 指向的地址是页面对齐的。
5. `pagesize * 2`：
   - 计算两个页面的字节数。假设页面大小为4096字节，那么 `pagesize * 2` 是8192字节。
6. `+ pagesize * 2`：
   - 将对齐后的地址向上偏移两个页面，最终得到的地址是距离起始地址两个页面的距离。
7. `(char *)`：
   - 将最终的计算结果转换回 `char*` 类型，以便用于字符指针的操作。

1. **页面大小**：
   - 在现代计算机系统中，内存是以页面为单位进行管理的，常见的页面大小为4096字节（4KB）。
   - 这意味着内存地址应该是页面大小的整数倍，以确保每个页面的起始地址是有效的。
2. **掩码的生成**：
   - 计算 `pagesize - 1` 生成一个掩码。例如，如果 `pagesize` 是4096字节，那么 `pagesize - 1` 的二进制表示是 `0xFFF`（111111111111，12个1）。
   - 取反后，掩码将变为 `0xFFFFF000`（在32位系统中，后12位为0，其余为1）。
3. **按位与操作**：
   - 将 `_mem`（假设其值为某个地址）与掩码进行按位与操作。这个操作会清除 `_mem` 的低12位（假设页面大小为4096字节，12位表示低于4096的地址）。
   - 例如，如果 `_mem` 是 `0x12345678`，那么：
     - `pagesize - 1` 是 `0x00000FFF`
     - `~(pagesize - 1)` 是 `0xFFFFF000`
     - 计算：`0x12345678 & 0xFFFFF000` 结果为 `0x12345000`。
4. **确保对齐**：
   - 经过这个操作，得到的结果是一个页面对齐的地址，即其低位部分为0。
   - 这确保了地址是页面大小的整数倍，因此可以安全地用作内存页的起始地址。

```c
void cache_decode_pretty(char *leaked, int index) {
  for(int i = 0; i < 256; i++) {
    int mix_i = ((i * 167) + 13) & 255; // avoid prefetcher
    if(flush_reload(mem + mix_i * pagesize)) {
      if((mix_i >= 'A' && mix_i <= 'Z') && leaked[index] == ' ') {
        leaked[index] = mix_i;
        printf("\x1b[33m%s\x1b[0m\r", leaked);
      }
      fflush(stdout);
      sched_yield();
    }
  }
}
```