

## Home Assignment 1

Due date: **November 27th, 23:59** (submit via moodle)

### Submission instructions

- **In this assignment, you can submit either in pairs or individually.** Even if working in a pair, each member should submit the assignment through the moodle.
- Submit one PDF (minimum 11 pt font) and one code file (**not** together in a ZIP).
- A solution submitted  $t$  seconds late will have its grade multiplied by  $1 - \left(\frac{t}{60 * 60 * 24 * 7}\right)^4$ .

## 1 Speculation

In this assignment you will try to make a new speculative execution attack work, and report your findings.

### 1.1 Background

**Store/load forwarding** When a processor executes a load from physical address  $A$ , it checks if the ROB contains a previous (“older”) store to the same physical address. If this is the case, the load does not need to read from the cache. Instead, the data written by the latest such store that precedes the load is *forwarded* to the load.

**Proposed attack: exploiting 4K-aliasing** While reading an Intel manual (<https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf>), Herbert H. Hacker discovers Section 15.8 that describes *4K aliasing*. This section says that if we have two virtual addresses  $A$  and  $B$  such that  $A = B \bmod 4096$  (referred to as *4K aliasing*) then a load to  $A$  that appears shortly in the code after a store to  $B$  has a performance penalty.

Herbert hypothesizes that this occurs because of speculative store/load forwarding. Consider that the addresses of stores and loads in the code are *virtual*, but store/load forwarding can occur only for accesses to the same *physical* address. However, performing the virtual to physical mapping takes time (accessing the TLB and maybe also page tables). Herbert’s theory is that to make progress without waiting for the translation, the processor uses the fact that the virtual memory page size is 4K (4096 bytes) and makes the store/load forwarding decision based only on the *lower 12 bits* of the virtual address (i.e., the offset in the page). If two addresses 4K-alias, the processor speculates that they will map to the same physical address and speculatively does store/load forwarding. (If the addresses do not 4K-alias, they cannot match after the translation, because the page size is 4K.) Later on, after the virtual to physical translation completes, the processor validates whether there was indeed a physical address match. If not, it will squash the load (which is now known to have been forwarded invalid data) and re-execute it.

This kind of speculation can be exploited in a *same-thread* attack, for example in the context of JavaScript or a similar VM. Suppose the VM writes to address  $A$ , which is secret and cannot be accessed by untrusted code. But if the untrusted code has a load to address  $B$  that 4K aliases with  $A$ , it will get the contents of the store through speculative forwarding and can hopefully leak it through a cache side-channel before being squashed.

## 1.2 Assignment

Your assignment is to implement Herbert's attack and try getting it to work. That is, to successfully leak through a side-channel some data that was speculatively forwarded because of a 4K alias. If the attack does not work, suggest an explanation to why it does not work. (An explanation requires pointing to a flaw in the reasoning leading to the attack, not hand-waving about timing problems or something similar.)

To help you, we provide in the attached `spectre.c` file a demonstration of the Spectre attack. Make sure the computer you work on is vulnerable to Spectre. To check for the vulnerability, compile `spectre.c` using `gcc -O0 -std=gnu99` (note the `-O0`) and run it. If the machine is vulnerable, you should see a printout like the following:

```
Reading 23 bytes:
Reading at malicious_x = 0xfffffffffffffa38... Success: 0x54=T score=2
Reading at malicious_x = 0xfffffffffffffa39... Success: 0x68=h score=2
Reading at malicious_x = 0xfffffffffffffa3a... Success: 0x65=e score=2
Reading at malicious_x = 0xfffffffffffffa3b... Success: 0x20== score=2
Reading at malicious_x = 0xfffffffffffffa3c... Success: 0x70=p score=2
Reading at malicious_x = 0xfffffffffffffa3d... Success: 0x61=a score=2
Reading at malicious_x = 0xfffffffffffffa3e... Success: 0x73=s score=2
Reading at malicious_x = 0xfffffffffffffa3f... Success: 0x73=s score=2
Reading at malicious_x = 0xfffffffffffffa40... Success: 0x77=w score=2
...
```

If you cannot find a vulnerable computer, you can work on the TAU machine `rack-mad-03`, which is vulnerable.

Read and understand `spectre.c` to see how to flush cache lines and leak bytes over a cache side-channel. You can then modify the file to implement Herbert's attack.

You can use the Linux tool `perf` to monitor CPU performance counter events of your program. You can read about `perf` here: <http://www.brendangregg.com/perf.html#CPUstatistics>. For example, on the TAU `rack-mad-03` machine, you can collect statistics on the 4K aliasing events by using the following command:

```
perf stat -e cpu/event=0x7,umask=0x1,name=ld_blocks_partial_address_alias/
```

## 1.3 What to submit

Submit two files: (1) Write up whether the attack works or not, and if not, your hypothesis for why not. (2) Your code (a single C file).