ECE 571 – Advanced Microprocessor-Based Design Lecture 15b

Vince Weaver

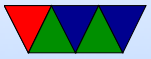
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

20 October 2022

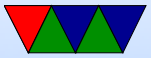
Announcements

- Midterm exam Thursday the 27th



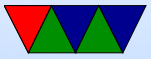
Computer Architecture Security

- We've made fast chips, but at what cost?



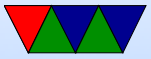
Side Channel Attacks

- Leak info in unexpected ways
- Timing attacks... code takes different number of instructions to execute either side of if/then. If encrypting can maybe tell difference between 0 and 1 unless each way takes exact same cycles
- Leakage: time, performance counters, RF noise, anything shared (caches, stalls on hyperthreads), LEDs on routers, etc

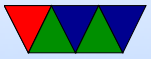


Meltdown

- `https://meltdownattack.com/`
- Problem: speculative execution can load data into caches even if you don't have VM permissions to do so
- Since by default most OSes map kernel and physical mem into VM address space, in theory you can read out *all* of memory
- This primarily an Intel bug, all chips with speculative execution, dating back to Pentium Pro?
- Some very high-end ARM (Cortex A75) too as well as

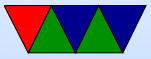


IBM Power? *Not* AMD though.



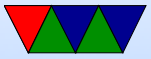
Cache Side-Channel Attacks

- Just loading speculative vaues into cache shouldn't be a problem as you can't actually read the values
- UNLESS there is some sort of side-channel
- If you can use the speculative data as an index to a memory load, you can bring yet another cache line in, the line depending on the value you shouldn't know.
- If you can determine if these lines are in cache you can know the content of the data.

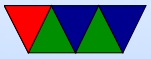


Determining if a Cache line is in Cache

- Evict and Time – run and time. Then evict just one line, then run again. If ran slower, it depended on data in that line
- Prime and Probe – load cache full of your data. Wait until code of interest runs. Then see how many of cache lines got kicked out.
- Flush and Reload
  - Single cache line granularity
  - Use `cflush` or similar to kick out a cache line



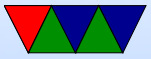
- Reload and time it, can tell if someone else had reloaded it in the meantime by how fast it loads



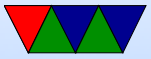
Meltdown – Toy Code

```
raise_exception()
access(probe_array[data*4096])
```

- The access should never happen, as the exception (segfault, etc) will trigger
- If exception slow enough, the access will likely be speculatively executed
- In theory the results of the access are thrown out so you cannot know it happened
- The speculative load might end up in the cache though, and you can probe to see if it did

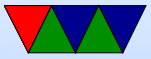


- The *bug* is that on Intel chips you can speculatively access kernel data from userspace and it will be cached despite the permission mismatch.
- Why multiply by 4096? Spread across multiple pages so the prefetcher doesn't get in the way.



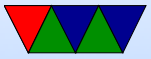
Performance

- Up to 500K/s read out



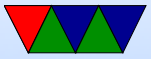
Meltdown – Issues

- Exception Handling – either a signal handler, or else forking a child to cause the exception
- Exception suppression transactional memory
- Limitation: Can get false zeros. Repeat until sure.



Workarounds

- Hardware
  - Turn off out-of-order (not possible, expensive)
  - Not allow user speculation to access kernel addresses
  - Not put data into cache until permission check completes
- Software
  - KASLR (Kernel Address Space Layout Randomization). If want arbitrarily read out kernel info need to find kernel

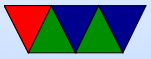


Turns out that with 40 bit address space and large physical memory (8GB) it doesn't take too many tries to find kernel

- KAISER (KPTI) – map kernel in separate address space.

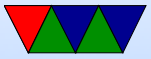
  Large overhead on switch in/out of kernel (syscalls, context switch)

  Up to 30% on someworkloads, almost none on others
- PCID (intel's ASID implementation on Westmere or newer) helps avoid complete TLB flush



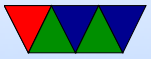
Spectre Vulnerability

- Unlike Meltdown, pretty much *any* processor with speculative execution affected
- Doesn't leak info from kernel, but from one part of program to another
- Why a problem? Well if javascript can read anything in rest of browser (passwords, history, etc)
- SPECulative execution, will haunt us for ages



Spectre – Depends on Branch Predictor

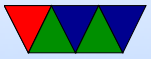
- You can reliably train branch predictor to hit/miss
- You can find if something is in the cache via timing
- Find a place in program where if it branches the wrong way it accesses a value of interest
- Manipulate branch predictor so it always predicts this taken
- Then go with an invalid value, but the predictor is trained to try
- Time analysis to get results



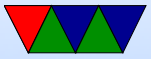
Spectre Variant 1 – Bounds Check

```
if (x<array1_size)
    y = array2[array1[x]*256];
```

- Ideally finds this code already existing in user code
- If mispredicts the check, will speculatively access the out-of-bounds value
- Attacker controls X
- Attacker trains the branch predictor that value is true with lots of runs
- Then passes in a value that is wrong but branch is predicted the previous way.

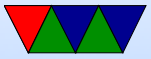


- array1_size is not cached, so it stalls and execution goes beyond
- Probe the cache much like meltdown



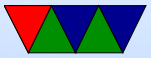
Indirect Branches

- Instead of relying on user code, train up the BTB
- Doesn't have to be the same address space, just has to alias in the BTB
- On many machines only 30 or fewer bits of BTB used to index

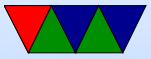


Spectre Variant 2 – Branch Target Injection

- Note for next year: review how this works
- X maliciously chosen
- Branch prediction manipulated to predict wrong
- arrays all kicked out of memory
- array1size was kicked out of RAM, so cache miss and slowly get value for RAM
- meanwhile predicts branch is good and so fetches array2[k*256]
- Eventually figures out and squashes wrong branch, but

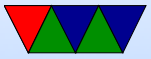


the fetch already underway into cache



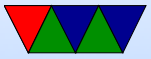
Finding a gadget

- Need to find code that runs with adversarial values are in register
- Not hard, often unused values leak across function calls (if a function doesn't use them)
- Need to find way to trigger a branch in a way that acts on these as pointers.
- Then find existing indirect jump
- Train the BTB to want to jump to our gadget
- clear out cache, perform attack



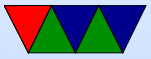
Notes

- some i7 up to 188 instructions can execute speculatively between
- can be triggered from Javascript. No clflush, but can evict all of cache by reading through an array.
- branch predictors on cpus are independent?



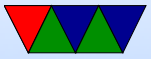
Workarounds

- Software
  - Disabe hires timers in javascript
  - Memory barriers – can halt speculation with special instructions, but have to insert them all through code where it might be an issue.
  - Kaiser/KPTI not help
  - Retpoline and IBRS, see next slides



Mitigation: New barriers

- Added by Intel with firmware update, new MSR
- IBRS – indirect branch restricted speculation
  flush branch predicto on entry to kernel, disable brpred on hyperthread
- STIBP – single-thread indirect branch prediction – disable brpred on sibling thread (currently they share brpred)
- IBPB – indirect br pred barrier – flush branch predictor state



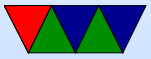
Mitigation: retpoline

- Indirect branches can be used for spectre attack
- Can you (at a performance cost) change all indirect branches to disable branch prediction?
- Original indirect call

```
jmp *%r11
```

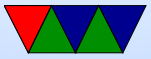
- Replace with this:

```
    call    set_up_target   ; skip ahead (return addr on
capture_spec:
    pause; lfence           ; trap to catch speculation
    jmp capture_spec        ; can't␣speculate␣out
```



```
set_up_target:
␣␣␣␣mov␣%r11,␣(%rsp)␣␣␣␣;␣overwrite␣return␣address␣with␣
␣␣␣␣ret␣␣␣␣␣␣␣␣␣␣;␣call␣using␣ret␣(confuse␣brpred)
```

- Return trampoline
- Convert indirect branch into a `ret` in a common location, makes it hard to train branch predictor.
- Also adds a code-trap so that if code speculates past the branch it gets trapped in an infinite loop
- Downside: all indirect branches now slower retpoline.



Are you vulnerable?

- On Linux look in `/proc/cpuinfo`
- Also can look in `/sys/devices/system/cpu/vulnerabil`