Micro-architectural usage aware Scheduling-Report

by Abhishek Santhanam(CS18B049) and R Raghu Raman(CS18B040)

Introduction

The problem that we are trying to solve is to build a scheduler which would detect micro-architectural attacks such as Spectre, Meltdown and Rowhammer and schedule them appropriately. We do this in multiple steps.

1) We first identify some specific parameters which can help us identify if a process is malicious, obtained from analyzing the statistics provided by running perf on some collection of processes.

2) We use this to make a deterministic heuristic in the scheduler, to identify in real time if a process is behaving maliciously.

3) After identifying that a process is an attack, it's priority in the wait-queue is decreased by increasing the vruntime.

4) This ensures that the process which is predicted as an attack eventually slows down and dies out.

5) To handle cases where a non-malicious process is detected as an attack we also decrease the increment in vruntime for the process if it's an incorrect prediction.

6) This detection and change in priority is done for every N(32 in our case) context switches. We have also analysed the performance for other values of N.

Concepts

1) Each of these attacks take advantage of a few loopholes in the micro-architecture.

2) Meltdown uses the loophole of out-of-order execution and tries to access the data in the cache which is not deleted after a context switch.

a. Consider the following code:

```
1. raise_exception();
2. //the line below is never reached
3. access(probe_array[data * 4096]);
```

If instruction 3 is executed out of order before the kernel terminates then the data that it tries to access though not stored in the memory or registers would be stored in the cache and this is what the attack essentially tries to access and leak.

b. This shows that there would be a lot of accesses to the cache and considerable amount of them would be misses, hence in the perf-stat we used LLC-Cache misses as a parameter for the heuristics.

3) Spectre exploits branch prediction to bypass the isolation between user level processes, if trains the branch predictor to take the conditional branch and then provides an out-of-bound exception. Even in this the data that can be leaked is stored in the Cache and hence leads to a lot of accesses to cache enabling us to again use the LLC-Cache misses as a parameter for the heuristics.

4) Perf which gives us hardware related statistics about various processes also aligns with the same. We see that normal processes like the ones found in MIbench have on average around 20% LLC-Misses whereas the attacks have more than 80% LLC-Misses.

Code

The changes to the kernel are as follows:

- New attributes have been added to the kernel PCB located in include/linux/sched.h **task_struct**. These are to store the number of rounds of context switches elapsed, an array of the values of the events and the weight to penalized attacks with a higher vruntime. These attributes are initialized in include/linux/init_task.h for the init process and kernel/fork.c **copy_process** for other processes.
- Heuristics have been added to update hpcs and identify malware in kernel/sched/fair.c **update_curr**. Some auxiliary functions have been added above the same.
- The number of rounds is a variable that can be changed at runtime of the kernel. This is declared in kernel/sched/fair.c and interfaced in kernel/sysctl.c and include/sched/sysctl.h. This can be changed at runtime using **/proc/sys/kernel/maa_sched_rounds** which is a temporary file. This parameter takes a value between 1 and 32, and 0 to indicate that the attack detection feature is turned off.

Testing

- All testing was done using real time got from running the test processes along with the linux time command.
- MIBench: 13 programs from the MIBench collection of benchmarks we used for the test. The test included compile time, running the large version of the benchmark 10 times, and then clean up time. The total time is used.
- Meltdown: The time taken to run meltdown attack (only the ./kaslr) for a significant amount of time. (This attack is an infinite loop)
- Rowhammer: The time taken to execute 15 iterations of the rowhammer attack.

Results

N \ Process	MIBench	Meltdown	Rowhammer
None(inf)	28.347s	28.026s	18.874s
2	27.552s	42.896s	18.041
8	26.908s	37.840s	17.506s
32	27.612s	36.600s	17.908s

We see that an attack like meltdown is significantly slowed down even with a simple heuristic. An attack like rowhammer which does not produce many cache misses is not slowed down by much. In the future having a better heuristic can help detect these attacks better.

Contribution

- Theory regarding micro-architechture attacks and mitigation - Abhishek Santhanam
- Analysis of events and testing using Hardware performance counters- Raghu Raman
- Kernel Code - Both
- Report - Abhishek Santhanam
- Demonstration - Raghu Raman

Reference

1. Spectre and Meltdown Attacks Detection using Machine Learning and Hardware Performance Counters.[1]

2. Cache Side Channel Attacks: Exploitability and Countermeasures by Gorka Irazoqui and Xiaofei.[2]

3. A survey on Micro architectural Timing Attacks and Countermeasures on Contemporary Hardware.[3]

4. CPU Hardware Performance Counters for Security by Nishad Herath and Anders Fogh. [4]

5. Predicting program phases and Defending against Side-Channel Attacks using Hardware Performance Counters. [5]

6. Linux code used from [6].

7. Intel Software Developer's Manual Part 3 [7].

8. Discussion on how to use rdpmc, wrmsr and rdmsr [8].

9. MIBench [9].

10. Rowhammer [10].

11. Meltdown [11].