Anamoly & Malware Detection

CS3500 COURSE PROJECT

Team Members:

D Mani Kireeti CS18B014

Prashasth R B CS18B039

Project Goal:

Designing a scheduler that can detect and handle malware & modify the Linux scheduler to detect anomalies in selected processes.

Contribution:

Work was divided evenly among both the team members. Research, Coding, Ideas was everything pursued together.

Heuristics:

For the problem of Anamaly Detection, we decided our main factor to differentiate anomalies from normal processes is Branch Miss Predictions. To keep it uniform throughout and maintain regularised values, we updated the main factor to (Branch Miss Predictions / Time Elapsed).

For obtaining Branch Miss Predictions, we are using Model Specific Registers (MSR's). With the introduction of the Pentium processor, Intel provided a pair of instructions (RDMSR and WRMSR) to access current and future "model-specific registers". Reading and writing to these registers is handled by the rdmsr and wrmsr instructions, respectively. As these are privileged instructions, they can be executed only by the operating system.

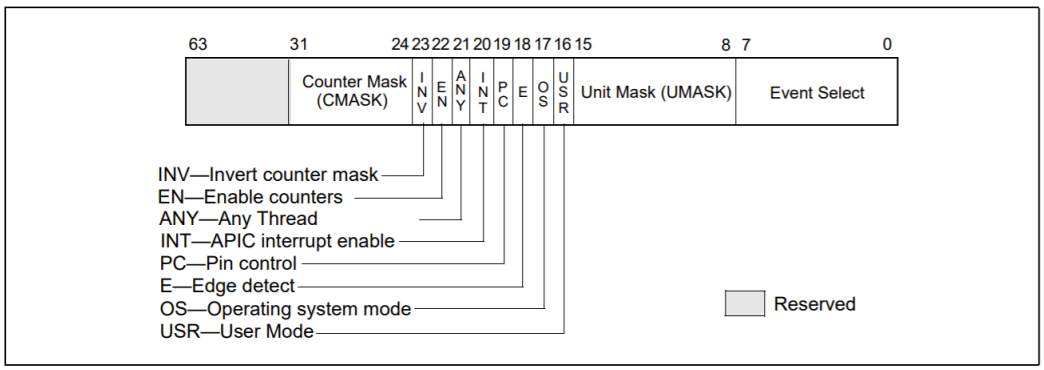

Figure 18-6. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

The above figure represents a typical Model Specific Register. With different combinations of UMASK and Event Select values, different performance counters are achieved.

Table 18-1. UMask and Event Select Encodings for Pre-Defined Architectural Performance Events

5	Branch Instruction Retired	00H	C4H
6	Branch Misses Retired	00H	C5H

As our intended performance counter is Branch Prediction Misses, We set UMask to 00H (0), Event Select to C5H (197), Operating System mode to 1 and Enable events to 1.

Initially we need to enable the PMC0 bit from IA32_PERF_GLOBAL_CTRL MSR so that all the performance counter bits are enabled. Now, we configure IA32_PERFEVTSEL0 as mentioned above. We update the Branch Miss Predictions in a free register (decimal address:193), using it as a counter.

We use rdmsr to read from the above counter register which then returns a integer. We maintain a field in task_struct namely task_br_misp which is updated using the above returned integer. This is executed right after the process is executed. For Debugging purposes, we print the task->pid(pid of process) along with the tsk->task_br_misp(Branch Miss Predictions) to the kernel long, which can be obatined by “demsg” command in terminal.

We obtain the time elapsed of a process by using task structure properties tsk->se.sum_exec_runtime

We obtain the main factor by dividing the Branch Miss Predictions (task_br_misp) with time elapsed by process (tsk->se.sum_exec_runtime)

For Comparision purposes we check weather this factor is far greater than an average factor (This average factor is obtained by dividing the total branch miss predictions with total time elapsed until this process). If it is greater, we assume it to be a anamoly and we increase the static priority(tsk->static_prio)

of that process by 1. By this we can asssure that an Anamoly's will keep increasing.

Change Log:

Linux-5.4.82/kernel/sched/core.c

```
55   unsigned long long int total_br_misp = 0;
56   unsigned long long int total_exec_time = 0;
```

```
4091  static void __sched notrace __schedule(bool preempt)
4092  {
4093          long long int llo=0,lho=0;
```

```
4142            next = pick_next_task(rq, prev, &rf);
4143            clear_tsk_need_resched(prev);
4144            clear_preempt_need_resched();
4145            wrmsr(911,1,0);
4146            llo |= 197;
4147            llo |= (1<<16);
4148            llo |= (1<<21);
4149            llo |= (1<<22);
4150            wrmsr(390,llo,lho);
4151            wrmsr(193,0,0);
```

```
4240  asmlinkage __visible void __sched schedule(void)
4241  {
4242          long long int llo,lho;
4243          struct task_struct *tsk = current;
4244          rdmsr(193,llo,lho);
4245          tsk->task_br_misp += (llo + (lho << 32));
4246          total_br_misp+=tsk->task_br_misp;
4247          total_exec_time+=tsk->se.sum_exec_runtime-tsk->se.prev_sum_exec_runtime;
4248          if(total_exec_time != 0)
4249          {
4250                  if(tsk->task_br_misp/tsk->se.sum_exec_runtime >= 100 * (total_br_misp/total_exec_time))
4251                  {
4252                          if(tsk->static_prio <= 138)
4253                          tsk->static_prio += 1;
4254                  }
4255          }
4256          printk("Branch misses by %d is %d\n",tsk->pid,tsk->task_br_misp);
```

Linux-5.4.82/kernel/fork.c

->long _do_fork(struct kernel_clone_args *args)

```
2379            p->task_br_misp=0;
```

Linux-5.4.82/include/linux/sched.h

```
624   struct task_struct {
625            unsigned int task_br_misp;
```

References:

- https://www.intel.in/content/www/in/en/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html (18th chapter)
- https://en.wikipedia.org/wiki/Model-specific_register
- https://elixir.bootlin.com/linux/latest/source
- https://stackoverflow.com/
- https://man7.org/linux/man-pages/man2/perf_event_open.2.html

Note:

Despite our best efforts, we were not able to compile the linux kernel in either of our laptops and desktop. We tried dual boot, virtual box, vmware player 16, vmware workstation pro on ubuntu 16.04,20.04 and kali linux, but nothing worked out. Finally, we were only able to compile the kernel without any errors.