Review of paper

Mondrix: Memory Isolation for Linux using Mondriaan Memory Protection

by E. Witchel, J. Rhee, K. Asanović

1 Summary

The paper demonstrates the benefits of small-grain memory protection, as provided by MMP, by porting Linux to an MMP architecture. The scheme is used to isolate kernel modules in their own protection domains, and the beneficial effects on software robustness are evaluated. It specifically demonstrates the benefit of a fast, hardware-supported protection-domain (PD) switch.

2 Discussion of approach

Linux was split into a total of 11 protection domains, most of them device drivers or sub-modules of device drivers. It seems that the system was configured with a minimum of drivers for this purpose, but that is not explicitly stated.

Two of the protection domains contain new (privileged) code (called "supervisor") that manages the MMP data structures. All other domains are subject to MMP access restrictions (although still operating in kernel mode). Significant work was required to introduce the correct switch gates (points of PD change) and supervisor calls by which owners of data grant sharing rights to other PDs. While the paper states that only 2kloc of Linux kernel code needed to be changed, this is likely a very large effort as I suspect that those 2000 lines were spread throughout the kernel. Furthermore, the authors only ported a small number of drivers (and reasonably so) — extending this to all of Linux seems like a task of herculean proportions.

The authors made a number of improvements to MMP in the process of this work, which shows the importance of evaluating a new architecture on real systems.

3 Evaluation

The approach is evaluated in terms of function and performance. The functional evaluation identifies an actual kernel bug (which seems low, given the sorry state of driver code, and in fact the abstract mentions two bugs!)

Fault injection is then used to show that Mondrix more frequently than Linux detects bugs before they result in irrecoverable damage. I find such fault injection experiments (although used in a number of recent papers) quite weak, as they do not seem to produce a reasonable approximation of real-life OS bugs. Kernel bugs tend to be conceptual (violation of internal protocols such as incorrect synchronisation), breakage as a result of changes in other code, off-by-one errors etc. The kind of random errors introduced by the fault injection scheme would mostly be short-lived in a real kernel.

The performance evaluation uses system simulators to estimate the overheads of MMP in terms of extra instructions executed or extra memory stalls. Also measured are the number of cross-domain calls and the memory overhead, and the extra traffic caused by protection-lookaside buffer (PLB) reloads.

I find most of those measures rather meaningless. The most useful data are the extra instructions and memory stalls, although they are based on the (for x86) highly simplistic assumption of one instruction executed per clock cycle.

The data on memory overhead, expressed as the relative reduction in free kernel memory at the end of the runs, is highly misleading. The systems under test ran for relatively small amounts of time (seconds or minutes), and most were essentially single-process loads. Under those circumstances it can be assumed that the kernel's memory pool is vastly excessive, and a 10% reduction in free memory means very little (and could actually be quite large).

Similarly, the insignificant PLB refill traffic is totally expected: given the very small number of PDs, the PLB working set would be much smaller than PLB capacity, so there is no reason to expect PLB misses at all. Incidentally, the paper does not state the size of the PLB (or any of the supervisor data structures, making independent confirmation of the results impossible). Were the reviewers sleeping?

More seriously, the paper makes little attempt at actually analysing the benefits of various aspects of MMP. It seems that the fast protection-domain switch is beneficial for enabling the use of smaller protection units inside the kernel. However, this is not clearly established, as no attempt is made to compare the performance with e.g. Nooks. Such an evaluation would have been easy, by increasing cross-domain call costs in the simulator to values representative of x86, an architecture known for expensive context switches. Furthermore, a comparison with an existing architecture that supports fast context switches would have been appropriate. Itanium is such an architecture, supporting single-cycle mode switches and protection-domain switches in a few dozen cycles. Without such a comparison, the real value of MMP for supporting more fine-grained protection domains has not been established.

The case for fine-grained *memory protection* has been made even less. While the paper describes the use of MMP for protection individual stack frames (even words within a frame) and network packets, it is not clear to which degree this is actually used, and what its cost and benefit is. What *is* said is that in the slab allocator speed is traded for isolation, so there are at least some cases where there are perceived performance problems.

There is also absolutely no attempt at evaluating the scalability of MMP to larger numbers of PDs. At the least one would have expected a benchmark using a larger number of user processes (eg. the kernel compile which tends to be used as a standard benchmark in the Linux world). Such a benchmark, much more representative of real-life use scenarios, would have significantly increased cache contention and would likely increase Mondrix overheads.

A thorough cost-benefit analysis of MMP would seem absolutely essential in order to make the case for such a massive architectural change (and the immense followup cost from utilising it in the OS). The paper fails to do this, and replaces it with the optimistic statement that "the recent introduction of the NX bit to the x86 architecture [...] indicates that manufacturers are willing to add compatible hardware features to improve software robustness." I fail to see how the addition of such a simple, yet highly effective protection measure after decades of successful stack-smashing attacks gives raise to any optimism about the willingness of manufacturers to consider a very significant extension of the architecture. Certainly not without a very strong analysis of a kind which this paper does not present.

3.1 Open questions

- If not everything could be run in Simics, why not run everything in Bochs? Why not use a decent system simulator, such as M5?
- why does the system still have a kernel mode? This should really not be necessary. Is it only for making the port easier? This should be explained.
- Mondrix is really a multi-server system running on something like a microkernel. Why not say so, and compare to other microkernel systems?
- why didn't the authors submit a patch for the kernel bug they found?

4 Overall impression

While the Mondrix implementation is an amazing achievement, I'm uncertain what we really have learnt from the paper. The introduction claims five contributions:

- design and implementation of a fine-grained kernel memory protection system

  Ok, the authors implemented the software support required for MMP, which demonstrates that this can be done. This proves that the software support for MMP can be implemented in a few 1000 loc.

- implementation of a compartmentalised Linux

  This is essentially what Nooks did. It is not clear that Mondrix is significantly finer-grained (with only 11 PDs)

- modifications to MMP

  The main addition seems to be improvements to the cross-domain-invocation mechanism, specifically a separate lookup cache. This is only relevant as far as MMP itself is relevant

- evaluation of performance and space overheads

  I have argued above that this evaluation is unconvincing

- fault-injection experiments showing how Mondrix can catch errors

  That fact is hardly surprising, but I expressed my scepticism about the value of those experiments.

So what we learnt is that an MMP-based OS is possible, and it is even possible to retrofit MMP support into a legacy OS. In order to do this, MMP had to be improved first. That seems hardly earth-shaking.