# The current state of kernel page-table isolation

## [LWN subscriber-only content]

By **Jonathan Corbet**
December 20, 2017

At the end of October, the KAISER patch set was unveiled; this work separates the page tables used by the kernel from those belonging to user space in an attempt to address x86 processor bugs that can disclose the layout of the kernel to an attacker. Those patches have seen significant work in the weeks since their debut, but they appear to be approaching a final state. It seems like an appropriate time for another look.

This work has since been renamed to "kernel page-table isolation" or KPTI, but the objective remains the same: split the page tables, which are currently shared between user and kernel space, into two sets of tables, one for each side. This is a fundamental change to how the kernel's memory management works and is the sort of thing that one would ordinarily expect to see debated for years, especially given its associated performance impact. KPTI remains on the fast track, though. A set of preparatory patches was merged into the mainline after the 4.15-rc4 release — when only important fixes would ordinarily be allowed — and the rest seems destined for the 4.16 merge window. Many of the core kernel developers have clearly put a lot of time into this work, and Linus Torvalds is expecting it to be backported to the long-term stable kernels.

KPTI, in other words, has all the markings of a security patch being readied under pressure from a deadline. Just in case there are any smug ARM-based readers out there, it's worth noting that there is an equivalent patch set for arm64 in the works.

### 51 Patches and counting

As of this writing, the x86 patch series is at version 163. It contains 51 patches, so we can all be grateful that most of the intervening versions were not posted publicly. The initial patch set, posted by Dave Hansen, has been extensively reworked by Thomas Gleixner, Peter Zijlstra, Andy Lutomirski, and Hugh Dickins, with suggestions from many others. Any bugs that remain in this work won't be there as the result of a lack of experienced eyeballs on the code.

Page tables on contemporary systems are organized in a tree-like structure that makes for efficient storage of a sparse memory map and supports the huge pages feature; see this 2005 article for more details and a diagram of how it works. On a system with four levels of page tables (most largish systems, these days), the top level is the page global directory (PGD). Below that come the page upper directory (PUD), page middle directory (PMD), and page-table entries (PTE). Systems with five-level page tables insert a level (called the P4D) just below the PGD.

Page-fault resolution normally traverses this entire tree to find the PTE of interest, but huge pages can be represented by special entries at the higher levels. For example, a 2MB chunk of memory could be represented by either a single huge-page entry at the PMD level or a full page of single-page PTE entries.

In current kernels, each process has a single PGD; one of the first steps taken in the KPTI patch series is to create a second PGD. The original remains in use when the kernel is running; it maps the full address space. The second is made active (at the end of the patch series) when the process is running in user space. It points to the same directory hierarchy for pages belonging to the process

itself, but the portion describing kernel space (which sits at the high end of the virtual address space) is mostly absent.

Page-table entries contain permission bits describing how the memory they describe can be accessed; these bits are, naturally, set to prevent user space from accessing kernel pages, even though those pages are mapped into the address space. Unfortunately, a number of hardware-level bugs allow a user-space attacker to determine whether a given kernel-space address is mapped or not, regardless of whether any page mapped at that address is accessible. That information can be used to defeat kernel address-space layout randomization, making life much easier for a local attacker. The core idea behind KPTI is that switching to a PGD lacking a kernel-space mapping will defeat attacks based on these vulnerabilities, of which we have apparently not yet seen the last.

**Details**

The idea is simple but, as is so often the case, there are a number of troublesome details that turn a simple idea into a 51-part patch series. The first of those is that, if the processor responds to a hardware interrupt while running in user mode, the kernel code needed to deal with the interrupt will no longer exist in the address space. So there must be enough kernel code mapped in user mode to switch back to the kernel PGD and make the rest available. A similar situation exists for traps, non-maskable interrupts, and system calls. This code is small and can be isolated from the rest, but there are a number of tricky details involved in handling that switch safely and efficiently.

Another complication comes in the form of the x86 local descriptor table (LDT), which can be used to change how the user-space memory layout looks. It can be tweaked with the little-known `modify_ldt()` system call. The early POSIX threads implementation on Linux used the LDT to create a thread-local storage area, for example. On current Linux systems, the LDT is almost unused but some applications (Wine, for example) still need it. When it is used, the LDT must be available to both kernel and user space, but it must live in kernel space. The KPTI patch set shuffles kernel memory around to reserve an entire entry at the PGD level for the LDT; the space available for `vmalloc()` calls shrinks to a mere 12,800TB as a result. That allows space for a large number of LDTs, needed on systems with many CPUs. One result of this change is that the location of the LDT is fixed and known to user space — a potential problem, since the ability to overwrite the LDT is easily exploited to compromise the system as a whole. The final patch in the series maps the LDT read-only in an attempt to head off any such attacks.

Another potential vulnerability comes about if the kernel can ever be manipulated into returning to user space without switching back to the sanitized PGD. Since the kernel-space PGD also maps user-space memory, such an omission could go unnoticed for some time. The response here is to map the user-space portion of the virtual address space as non-executable in the kernel PGD. Should user space ever start running with the wrong page tables, it will immediately crash as a result.

Finally, while all existing x86 processors are seemingly affected by information-disclosure vulnerabilities, future processors may not be. KPTI comes with a measurable run-time cost, estimated at about 5%. That is a cost that some users may not want to pay, especially once they get newer processors that lack these problems. There will be a `nopti` command-line option to disable this mechanism at boot time. The patch series also adds a new "feature" flag (`X86_BUG_CPU_INSECURE`) to indicate vulnerable CPUs; it is set on all x86 CPUs currently, but might not be on future hardware. In the absence of this feature flag, page-table isolation will automatically be turned off.

Approximately one month remains before the opening of the 4.16 merge window. During that time, the KPTI patch set will undoubtedly go through a number of additional revisions as the inevitable glitches come to light. Once things settle down, though, it would appear that this code will be merged and backported to stable kernels in a relative hurry. Apparently, we can look forward to slower — but more secure — kernels as a new-year's present.

Post a comment

**The current state of kernel page-table isolation**
Posted Dec 20, 2017 16:06 UTC (Wed) by **luto** (subscriber, #39314) [Link]

> That allows space for a large number of LDTs, needed on systems with many CPUs

Not quite. The reserved space is per-process and contains at most two LDTs. I reserved all that space because the pagetable management for that space is more like user memory than kernel memory, and mixing the two styles in the same PGD entry could lead to nasty synchronization issues.

The reason that there are two LDTs per process is to keep atomic LDT switches simple. The old and new LDTs are both mapped and then all affected CPUs are notified of the change.

Reply to this comment

The current state of kernel page-table isolation (josh, Dec 20, 2017 18:43 UTC (Wed))
The current state of kernel page-table isolation (luto, Dec 20, 2017 19:59 UTC (Wed))
The current state of kernel page-table isolation (josh, Dec 20, 2017 20:06 UTC (Wed))
The current state of kernel page-table isolation (luto, Dec 22, 2017 17:57 UTC (Fri))
The current state of kernel page-table isolation (dvrabel, Dec 21, 2017 14:35 UTC (Thu))
The current state of kernel page-table isolation (luto, Dec 21, 2017 15:25 UTC (Thu))
The current state of kernel page-table isolation (roc, Dec 25, 2017 20:20 UTC (Mon))
The current state of kernel page-table isolation (zlynx, Dec 27, 2017 16:47 UTC (Wed))

**The current state of kernel page-table isolation**
Posted Dec 25, 2017 7:21 UTC (Mon) by **joib** (subscriber, #8541) [Link]

Is this similar to the 4G/4G patches of yore?

Reply to this comment

The current state of kernel page-table isolation (joib, Dec 25, 2017 7:44 UTC (Mon))

**The current state of kernel page-table isolation**
Posted Dec 25, 2017 16:09 UTC (Mon) by **joib** (subscriber, #8541) [Link]

The comments in the linked KAISER article mentioned that SPARC and s390 already use separate address spaces for user and kernel. Do those have some hw feature that allows them to do it with less overhead (if so, what?) , or is it just a convention?

And what about risc-v? Have they managed to avoid this or are they also vulnerable? I see that the privileged isa is still in draft status, so maybe they can still fix it?

Reply to this comment

The current state of kernel page-table isolation (farnz, Dec 26, 2017 19:04 UTC (Tue))
The current state of kernel page-table isolation (joib, Dec 26, 2017 19:53 UTC (Tue))

**The current state of kernel page-table isolation**
Posted Jan 2, 2018 20:29 UTC (Tue) by **mrhines** (subscriber, #119681) [Link]

On the issue of the potential 5% performance penalty, is there any thought being given to exception-less system calls as a solution to flushing all the time?
https://www.usenix.org/legacy/events/osdi10/tech/full_pap...

Reply to this comment