Zack's Kernel News

Chronicler Zack Brown reports on the latest news, views, dilemmas, and developments within the Linux kernel community.

By Zack Brown

Go Tell It On the Mou-ountain, 3.0 Is Heeeeeere

There has been much giddiness and consideration of the Ballmer Peak as a result of Linus Torvalds's recent announcement that a 3.0 kernel release was imminent.

Not everyone was thrilled about the version change. On learning that Linus did not intend to include any extreme changes in this release beyond the version number itself, Mustapha Rabiu remarked, "We applaud the fact that it'll be just as hideous as 2.6.x."

Actually, as it turns out, the 2.6 kernel has been in existence for so long that a kind of "Y2K" bug has crept into the kernel. A lot of scripts and other support code in the source tree has come to assume "2.6" as an eternal prefix, and so a lot of the patches going into the 3.0 release have to do specifically with fixing all the breakage from the version number update.

In fact, the switch to 3.0 also includes a significant change to the version numbering system itself. The 2.6 kernels all have a third number representing the actual release (2.6.1, 2.6.2, etc.), but the 3.0 kernel will just use the second number for releases (3.1, 3.2, etc.). The migration from a three-numbered version to a two-numbered version, besides requiring more fixes in the source tree, will also allow the folks maintaining stable kernel trees to abandon their four-numbered version system, which was pretty cumbersome and annoying, and take over the newly available third number (3.1.1, 3.1.2, 3.1.3, etc.). On hearing this news, Willy Tarreau, one of the stable kernel maintainers, expressed a big "THANK YOU" to Linus. At last, he said, stable kernel versions would no longer look quite so much like IP numbers.

Joe Pranevich, author of various "Wonderful World of Linux" documents describing the changes between major version changes, heard about the upcoming change and emerged from a seven-year hiatus (2.6 has been with us for quite a while) to begin summarizing all the changes that have happened in the intervening years. A bit later he released the fruits of those labors, at *http://www.kniggit.net/wwol30/*, and said that this URL would always contain the latest draft.

Among various responses to Joe's document was some concern that 3.0 represented both an immense change and a relatively minuscule one. Immense, because of everything that has gone into the 2.6 tree leading up to 3.0, but minuscule because the actual change from the final 2.6 release to the first 3.0 release will contain primarily just the version number and a few minor fixes. The 3.0 kernel doesn't introduce warp field generators, transporter technology, or a port to Data's positronic net. The consensus among the kernel folks reviewing Joe's early drafts was that he tone down his enthusiasm for all the accomplishments that were made during the 2.6 series because, after all, many of those features had by now been around for years.

Dell Discontinues Digests

Matt Domsch of Dell announced that the company would no longer produce digests of the linux-kernel mailing list nor the linux-scsi mailing list at *http://lists.us.dell.com/*. Hardware and software changes to their mailing list servers, he said, were the reason why.

Hiding the Kernel

Dan Rosenberg has produced a patch to randomize the location that the kernel will decompress into memory during boot. This patch is specifically targeted at attackers who try to exploit the addresses of known security vulnerabilities in the kernel. With this code, those attackers would no longer be able to rely on the location of their target in RAM.

Trying to do this, however, present many problems – many pitfalls in the dark recesses of the kernel. One question is where to find the entropy needed for random number generation. Not all hardware provides random number generation. Dan's first impulse, for systems that had no random number generation, was to simply do nothing. But Ingo Molnár suggested that a non-random solution could still be valuable, if it relied on numbers the attacker would not have access to, such as the RAM size or a BIOS signature (the value of a byte at a particular location in the BIOS).

Another issue is the entire 64-bit platform. It loads the kernel differently than on 32-bit systems and runs the kernel under a fixed virtual address mapping. To get the benefit of Dan's feature, 64-bit systems would have to change the way they did that mapping.

Another issue, as H. Peter Anvin pointed out, is that the bootloader would really be the most natural place for a feature of this kind. But because `grub` is the default bootloader on most systems and because Peter didn't think that any

ZACK BROWN

The Linux kernel mailing list comprises the core of Linux development activities. Traffic volumes are immense, often reaching 10,000 messages in a week, and keeping up to date with the entire scope of development is a virtually impossible task for one person. One of the few brave souls to take on this task is **Zack Brown**.

activity could ever be expected from the GRUB developers, he acknowledged that, in this case, doing the feature in-kernel might be the only way to actually get anything done.

Ingo also pointed out that kernel oops text might be harmed if the memory locations were randomized – how could anyone help debug a kernel panic if the RAM data meant nothing? He suggested converting the oops text automatically back to using canonical addresses at the time they were generated. It would be weird, but at least everyone's oopsen would be consistent.

A number of folks were also concerned with potential problems loading the kernel into just any old stretch of RAM. As coded by Dan, the kernel relocation would take place at a particular moment when other things, like initramfs, had already been loaded. Writing the kernel on top of any of those things (affectionately known as clobbering them) would result in an unusable system. And as H. Peter put it, "This is in fact an arbitrarily complex operation."

It was also, he said, one that could not be avoided by just picking a stretch of RAM based on some presumably safe heuristic. No, he said, the code would just have to work through the problem to find a legitimately safe stretch of RAM. But at least, he added, there was already existing code to tell exactly how big the stretch of RAM would have to be. The rest, however, would have to take the long way around.

Another issue, as Rafael J. Wysocki pointed out, was what Dan's code would mean for folks trying to hibernate their systems. As far as Rafael could see, Dan's initial implementation would break hibernation on both 32-bit and 64-bit systems. Over the course of discussion, it wasn't clear whether this would turn out to be a problem with Dan's code or with false assumptions made by the hibernation code. In particular, the question came up whether the hibernation code would allow a user to thaw out a system using a different kernel than was used to hibernate that system. In which case, Ingo said, it would be bad.

Every once in awhile, Linus Torvalds will come into a complex, difficult discussion and just solve it. So, as the number of devastating issues surrounding Dan's patch gradually began to clarify themselves, Linus stepped in with his own take on the situation.

First, he reframed the problem, saying that the patch would really only benefit Linux distributions, because any individual user compiling their own kernel was already loading an essentially internally randomized binary anyway.

So in that case, he proposed simply relinking a distribution kernel with some random little text offset. It wouldn't change the kernel at all, just shift it over by a little bit for each user. The linking could be done by the kernel installation scripts, so users wouldn't have to be aware of any change. It would just work.

There's often a lot of initial resistance when Linus does this, because he's essentially tossing out a large pile of work that people have put their hearts into. But in this case, Dan said, "given the number of non-trivial challenges that still need to be solved in order to implement load-time randomization, maybe this would be a better way forward."

With the doors blown off the hinges, other wild ideas started to emerge. At one point H. Peter said, "If we could modularize the core code we could have init code load the modules at all kinds of random addresses; they wouldn't even need to be contiguous in memory, and since we'd have full access to the memory layout at that point, we can randomize the [blank] out of *everything*." At roughly this point, Dan said he was putting his own patch on hold until folks could determine what the proper concept might be. And, he said it looked like H. Peter's idea might even be best. Dan said he wasn't shelving his project – he just wanted to hear more discussion before putting in more work in a particular direction. H. Peter meanwhile said he planned to implement his own idea in the syslinux bootloader, and the discussion petered out there.

It seems clear that everyone is unified in thinking that Dan's original goal of making kernel memory locations less predictable is important, but the specific design of that feature might turn out to be any number of things. My bet is that Linus's idea, as potentially the easiest to do, will be implemented for distributions, and probably H. Peter's idea will get further consideration and probably a lot of testing. But, it seems like Dan's idea of simply migrating the kernel from one part of memory to another involves too many complexities; especially given these relatively simple alternatives.