Kernel Articles at LWN (Paywall Just Expired)

By *Roy Schestowitz*
Created *14/11/2019 - 12:09am*
Submitted by Roy Schestowitz on Thursday 14th of November 2019 12:09:56 AM Filed under Linux [1]

- **Filesystem sandboxing with eBPF [2]**

  Bijlani is focused on a specific type of sandbox: a filesystem sandbox. The idea is to restrict access to sensitive data when running these untrusted programs. The rules would need to be dynamic as the restrictions might need to change based on the program being run. Some examples he gave were to restrict access to the ~/.ssh/id_rsa* files or to only allow access to files of a specific type (e.g. only *.pdf for a PDF reader).

  He went through some of the existing solutions to show why they did not solve his problem, comparing them on five attributes: allowing dynamic policies, usable by unprivileged users, providing fine-grained control, meeting the security needs for running untrusted code, and avoiding excessive performance overhead. Unix discretionary access control (DAC)?file permissions, essentially?is available to unprivileged users, but fails most of the other measures. Most importantly, it does not suffice to keep untrusted code from accessing files owned by the user running the code. SELinux mandatory access control (MAC) does check most of the boxes (as can be seen in the talk slides [PDF]), but is not available to unprivileged users.

  Namespaces (or chroot()) can be used to isolate filesystems and parts of filesystems, but cannot enforce security policies, he said. Using LD_PRELOAD to intercept calls to filesystem operations (e.g. open() or write()) is a way for unprivileged users to enforce dynamic policies, but it can be bypassed fairly easily. System calls can be invoked directly, rather than going through the library calls, or files can be mapped with mmap(), which will allow I/O to the files without making system calls. Similarly, ptrace() can be used, but it suffers from time-of-check-to-time-of-use (TOCTTOU) races, which would allow the security protections to be bypassed.

- **Generalizing address-space isolation [3]**

Linux systems have traditionally run with a single address space that is shared by user and kernel space. That changed with the advent of the Meltdown vulnerability, which forced the merging of kernel page-table isolation (KPTI) at the end of 2017. But, Mike Rapoport said during his 2019 Open Source Summit Europe talk, that may not be the end of the story for address-space isolation. There is a good case to be made for increasing the separation of address spaces, but implementing that may require some fundamental changes in how kernel memory management works.

Currently, Linux systems still use a single address space, at least when they are running in kernel mode. It is efficient and convenient to have everything visible, but there are security benefits to be had from splitting the address space apart. Memory that is not actually mapped is a lot harder for an attacker to get at. The first step in that direction was KPTI. It has performance costs, especially around transitions between user and kernel space, but there was no other option that would address the Meltdown problem. For many, that's all the address-space isolation they would like to see, but that hasn't stopped Rapoport from working to expand its use.

- **Identifying buggy patches with machine learning [4]**

The stable kernel releases are meant to contain as many important fixes as possible; to that end, the stable maintainers have been making use of a machine-learning system to identify patches that should be considered for a stable update. This exercise has had some success but, at the 2019 Open Source Summit Europe, Sasha Levin asked whether this process could be improved further. Might it be possible for a machine-learning system to identify patches that create bugs and intercept them, so that the fixes never become necessary?
Any kernel patch that fixes a bug, Levin began, should include a tag marking it for the stable updates. Relying on that tag turns out to miss a lot of important fixes, though. About 3-4% of the mainline patch stream was being marked, but the number of patches that should be put into the stable releases is closer to 20% of the total. Rather than try to get developers to mark more patches, he developed his machine-learning system to identify fixes in the mainline patch stream automatically and queue them for manual review.

This system uses a number of heuristics, he said. If the changelog contains language like "fixes" or "causes a panic", it's likely to be an important fix. Shorter patches tend to be candidates.

- **Next steps for kernel workflow improvement [5]**

The kernel project's email-based development process is well established and has some strong defenders, but it is also showing its age. At the 2019 Kernel Maintainers Summit, it became clear that the kernel's processes are much in need of updating, and that the maintainers are beginning to understand that. It is one thing, though, to establish goals for an improved

process; it is another to actually implement that process and convince developers to use it. At the 2019 Open Source Summit Europe, a group of 20 or so maintainers and developers met in the corner of a noisy exhibition hall to try to work out what some of the first steps in that direction might be.

The meeting was organized and led by Konstantin Ryabitsev, who is in charge of kernel.org (among other responsibilities) at the Linux Foundation (LF). Developing the kernel by emailing patches is suboptimal, he said, especially when it comes to dovetailing with continuous-integration (CI) processes, but it still works well for many kernel developers. Any new processes will have to coexist with the old, or they will not be adopted. There are, it seems, some resources at the LF that can be directed toward improving the kernel's development processes, especially if it is clear that this work is something that the community wants.

Linux

Links:
[1] http://www.tuxmachines.org/taxonomy/term/63
[2] https://lwn.net/Articles/803890/
[3] https://lwn.net/Articles/803823/
[4] https://lwn.net/Articles/803695/Identifying
[5] https://lwn.net/Articles/803619/