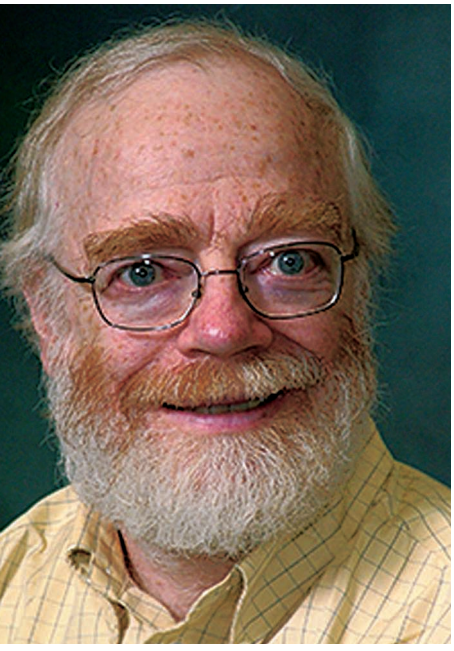

Steven M. Bellovin
Columbia University

Open Source and Trust

Of late, two major incidents involving open source software have been in the news. The more serious involved a Java logging package called "log4j"; the other involved a pair of JavaScript packages, "colors.js" and "faker.js". The incidents were very different, but there are some commonalities and some important distinctions in the lessons we should learn.

Log4j had a serious security hole, inserted intentionally by the developer who was unaware of the security implications. There was no malice involved, merely carelessness. But the error—effectively, trusting attacker-supplied input—would almost certainly have been spotted by any competent security person who looked carefully at its capabilities. The obvious corollaries to this are that neither the developers nor the many users of this package had the necessary security background. This kind of problem is not new; way back in 1994, Bill Cheswick and I wrote this about an Internet log analyzer: "Simple versions could easily fall victim to a sophisticated attacker who uses file names containing embedded shell commands."

The second issue was more curious. According to press reports, in apparent response to how the log4j developers handled the security reports—they worked hard, for free, through a holiday weekend to patch their code—the developer of colors.js and faker.js replaced the modules with intentionally broken versions. He had said, "Respectfully, I am no longer going to support Fortune 500s (and other smaller sized companies) with my free work… Take this as an opportunity to send me a six figure yearly contract or fork the project and have someone else work on it." Sites that pulled in the latest version fell victim to what is in essence a supply chain attack, much like the SolarWinds incident of a few months ago.

The wrong lesson to be drawn from these incidents is "don't trust open source software." Very reputable companies have had security holes about as egregious as log4j's; the SolarWinds attack showed that supply chain attacks can come from anywhere.

A better lesson is one we all learned in childhood: "know what you're putting in your mouth." That is, before you install a package such as log4j, know what you're getting. Actually examining the source code isn't easy—if it spotting security holes were that simple, they wouldn't exist at all: the developer would have found them. But understanding the capabilities of a package, e.g., the potential invocation of Java Naming and Directory Interface (JNDI) on untrusted input, should have been a red flag.

And that brings up the real lesson here: whom do you trust, and when do you make that decision?

The lesson applies to the newest versions of the colors.js and faker.js packages. "Install updates promptly" is a reasonable policy, but if you're running a production service you should always test new code before deploying it, even if you aren't worried about malice. I learned the lesson "Never install .0 of anything" about 50 years ago; the lesson still holds: new code is often buggy code, and you don't want to rely on it on your mission-critical servers.

And that brings up the real lesson here: whom do you trust, and when do you make that decision? If something is downloaded from an external source at run-time, you make the trust decision every time that program is run. More precisely, you have explicitly made the trust decision once, when you decided to invoke some external resource, but you make it again implicitly at each invocation. By contrast, with a resource that is downloaded once

Digital Object Identifier 10.1109/MSEC.2022.3142464
Date of current version: 21 March 2022

continued on p. 107

 Copublished by the IEEE Computer and Reliability Societies

Last Word *continued from p. 108*

and bundled with your code, you know when you've decided to rely on its correctness. You can (and often should) update to a newer version, but that gives you a chance to look at the changes and test them.

Open source from single developers can make this worse. If a major project or code base—MacOS, Windows, the Linux kernel, the Apache web server—contained deliberately sabotaged code, they may as well close down immediately; no one would ever trust them again. With small developers, you may not see that. More commonly, you may not see when economic or time pressures have made them stop working on a project, meaning that problems will never be fixed. What is needed is a larger framework in which these projects can be embedded, one that solves the economic, time pressure, and code testing issues. This will make it reasonable to extend trust implicitly, with confidence that nothing horrible will happen. ■

Steven M. Bellovin is a professor of computer science and affiliate law faculty at Columbia University. Contact him via https:// www.cs.columbia.edu/~smb.

Digital Object Identifier 10.1109/MSEC.2022.3152892