An anonymous coder nearly hacked a big chunk of the internet. How worried should we be?

April 4 2024, by Sigi Goode

Credit: Pixabay/CC0 Public Domain

Outside the world of open-source software, it's likely few people would have heard about XZ Utils, a small but widely used tool for data compression in Linux systems. But late last week, security experts uncovered a serious and deliberate flaw that could leave networked Linux computers susceptible to malicious attacks.

The flaw has since been confirmed as a critical issue that could allow a knowledgeable hacker to gain control over vulnerable Linux systems. Because Linux is used throughout the world in email and web servers and application platforms, this vulnerability could have given the attacker silent access to vital information held on computers throughout the world—potentially including the device you're using right now to read this.

Major software vulnerabilities, such as the SolarWinds hack and the Heartbleed bug, are nothing new—but this one is very different.

The XZ Utils hack attempt took advantage of the way open-source software development often works. Like many open-source projects, XZ Utils is a crucial and widely used tool—and it is maintained largely by a single volunteer, working in their spare time. This system has created huge benefits for the world in the form of free software, but it also carries unique risks.

Open source and XZ Utils

First of all, a brief refresher on open-source software. Most commercial software, such as the Windows operating system or the Instagram app, is "closed-source"—which means nobody except its creators can read or modify the source code. By contrast, with "open-source" software, the source code is openly available and people are free to do what they like

with it.

Open-source software is very common, particularly in the "nuts and bolts" of software which consumers don't see, and hugely valuable. One recent study estimated the total value of open source software in use today at US$8.8 trillion.

Until around two years ago, the XZ Utils project was maintained by a developer called Lasse Collin. Around that time, an account using the name Jia Tan submitted an improvement to the software.

Not long after, some previously unknown accounts popped up to report bugs and submit feature requests to Collin, putting pressure on him to take on a helper in maintaining the project. Jia Tan was the logical candidate.

Over the next two years, Jia Tan become more and more involved and, we now know, introduced a carefully hidden weapon into the software's source code.

The revised code secretly alters another piece of software, a ubiquitous network security tool called OpenSSH, so that it passes malicious code to a target system. As a result, a specific intruder will be able to run any code they like on the target machine.

The latest version of XZ Utils, containing the backdoor, was set to be included in popular Linux distributions and rolled out across the world. However, it was caught just in time when a Microsoft engineer investigated some minor memory irregularities on his system.

A rapid response

What does this incident mean for open-source software? Well, despite

initial appearances, it doesn't mean open-source software is insecure, unreliable or untrustworthy.

Because all the code is available for public scrutiny, developers around the world could rapidly begin analyzing the backdoor and the history of how it was implemented. These efforts could be documented, distributed and shared, and the specific malicious code fragments could be identified and removed.

A response on this scale would not have been possible with closed-source software.

An attacker would need to take a somewhat different approach to target a closed-source tool, perhaps by posing as a company employee for a long period and exploiting the weaknesses of the closed-source software production system (such as bureaucracy, hierarchy, unclear reporting lines and poor knowledge sharing).

However, if they did achieve such a backdoor in proprietary software, there would be no chance of large-scale, distributed code auditing.

Lessons to be learned

This case is a valuable opportunity to learn about weaknesses and vulnerabilities of a different sort.

First, it demonstrates the ease with which online relations between anonymous users and developers can become toxic. In fact, the attack depended on the normalization of these toxic interactions.

The social engineering part of the attack appears to have used anonymous "sockpuppet" accounts to guilt-trip and emotionally coerce the lead maintainer into accepting minor, seemingly innocuous code

additions over a period of years, pressuring them to cede development control to Jia Tan.

One user account complained, "You ignore the many patches bit rotting away on this mailing list. Right now you choke your repo."

When the developer professed mental health issues, another account chided, "I am sorry about your mental health issues, but its important to be aware of your own limits. "

Individually, such comments might appear innocuous, but in concert become a mob.

Credit: @glyph@mastodon.social

We need to help developers and maintainers better understand the human aspects of coding, and the social relationships that affect, underpin or dictate how distributed code is produced. There is much work to be done, particularly to improve the recognition of the importance of mental health.

A second lesson is the importance of recognizing "obfuscation", a process often used by hackers to make software code and processes difficult to understand or reverse-engineer. Many universities do not teach this as part of a standard software engineering course.

Third, some systems may still be running the dangerous versions of XZ Utils. Many popular smart devices (such as refrigerators, wearables and home automation tools) run on Linux. These devices often reach an age at which it is no longer financially viable for their manufacturers to update their software—meaning they do not receive patches for newly discovered security holes.

And finally, whoever is behind the attack—some have speculated it may be a state actor—has had free access to a variety of codebases over a two-year period, perpetrating a careful and patient deception. Even now, that adversary will be learning from how system administrators, Linux distribution producers and codebase maintainers are reacting to the attack.

Where to from here?

Code maintainers around the world are now thinking about their vulnerabilities at a strategic and tactical level. It is not only their code itself they will be worrying about, but also their code distribution mechanisms and software assembly processes.

My colleague David Lacey, who runs the not-for-profit cybersecurity organization IDCARE, often reminds me the situation facing cybersecurity professionals is well articulated by a statement from the IRA. In the wake of their unsuccessful bombing of the Brighton Grand Hotel in 1984, the terrorist organization chillingly claimed: "Today we were unlucky, but remember we only have to be lucky once. You will have to be lucky always."