Product

XZ-actly What You Need (CVE-2024-3094): Detecting Exploitation with Oligo

Guy Kaplan, Uri Katz, Nitzan Mousseri
April 2, 2024

Background

Security teams and developers lost sleep this weekend over the `XZ` backdoor discovered on March 29 by a sharp-eyed developer. While teams scrambled to make sure the impacted version wasn't part of their distro (yet), the lucky early discovery limited the damage.

For customers using Oligo, running a distro with the backdoor would have triggered an instant alert during an exploitation attempt—because the backdoor caused a library (`liblzma`) to behave in a way it hadn't behaved before.

We didn't have to write any new detection rules. Our existing product, in its existing state, could have detected *and stopped* the `XZ` backdoor before it did any damage.

We're not talking about vulnerability scanning (though we can do that, too).

This is about a one-of-a-kind feature of the Oligo platform: Oligo ADR uses unique library-level profiling technology that can detect exploitation attempts on running applications and OS packages instantly—even for unknown risks that have not yet been named or patched.

When `log4j` hit, we knew we had to find a solution, and that is why Oligo was founded.
Now, when `XZ` hits, we already have the solution.

How the XZ Backdoor (CVE-2024-3094) Actually Works

A thousand articles have been written about why the `XZ` backdoor was (and is) so bad, and which versions are impacted (5.6.0 and 5.6.1).

But how does it work in practice? What would an attack that used the `XZ` backdoor look like?

Researchers around the world have already figured out that attackers have utilized a user-level hook to divert the control flow of the `RSA_public_decrypt` function to instead utilize `libc.so system()` function to execute a payload provided by the attacker, essentially leading to remote code execution (RCE).

Key takeaway: once the backdoor is triggered, the control flows back to the `liblzma` library to execute the payload by triggering a call to `libc.so system()` function in order to run arbitrary commands on the system.

How Oligo ADR Detects Exploitation of CVE-2024-3094

Oligo has a huge library of runtime profiles for a wide range of libraries and OS packages — `liblzma` included. The Oligo Platform uses our vast knowledge base of expected library behaviors to identify when libraries behave in anomalous ways—which indicates an active exploit beginning or in progress.

We dug in to our `liblzma` data, and we noticed something amazing:

The prebuilt Oligo profile for the `liblzma` library shows that it was never observed calling the system function at any point during its runtime as part of its legitimate intended behavior.

This means that triggering the backdoor would be caught by Oligo and create an incident in the Oligo ADR platform automatically.

To illustrate how Oligo ADR detects this attack, we replicated it using the examples provided by Anthony Weems (amlweems), available at: https://github.com/amlweems/xzbot.git

Anthony successfully extracted the attackers' public key and substituted it with his own, allowing us to recreate the attack on our cluster.

Running `sshd` with the following command:

```
env -i LC_LANG=C LD_PRELOAD=/home/oligo/xzbot/liblzma.so.5.6.1.patch
/usr/sbin/sshd
```

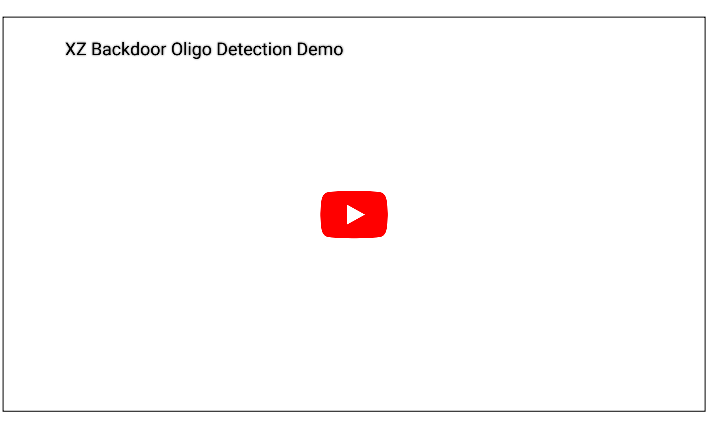
causes the backdoored `liblzma` to be loaded to the process, which also causes a hook to be placed on the `RSA_public_decrypt` function.

Once an ssh connection is made to the machine, the control flow diverges to the backdoored `liblzma` library, which in turn tries to decrypt the payload using the hard-coded key.

If the payload is signed with the correct key, the `system()` function is invoked, which is immediately detected by the eBPF-based Oligo sensor.

In the video below, watch how an attacker activates the backdoor to execute the `id` command and directs its output to the `/tmp/.xz` file.

The code is running on a Debian image inside a Kubernetes cluster which is monitored by the Oligo sensor.



The video shows how the Oligo sensor immediately recognizes the exploitation of the malicious backdoor by comparing the behavior of the `liblzma` library during exploitation against its normal behavior.

Detecting the "Undetectable" (Want to Try It Out?)

These facts mean the cat's out of the bag: none of our customers were impacted by the `XZ` backdoor. In fact, if any of them had been, they'd have been notified immediately—and our researchers could have been the first people to detect this supply chain vulnerability.

As the libraries with malicious code spread to more organizations, one of our customers would have updated to an impacted distro and the problem would have been identified. Because of this, some of the bigger disaster scenarios imagined by journalists and researchers could not have occurred—Oligo would have caught this issue long before it became part of everyone's distro.

We've heard a common refrain about the `XZ` backdoor: "no product could have detected it!". We'd like to politely disagree.

Generally the skepticism arises from the way it didn't look like a typical attack, and didn't have a CVE assigned. Because of the specific method of the attack, most security products today couldn't have caught this issue. EDR and container solutions were totally blind to it.

But Oligo is a different kind of product. The Oligo Platform can detect it—and other exploits that haven't yet been in the headlines..

Detecting tomorrow's zero days today? We know we're making big promises. We welcome you to test the Oligo Platform for yourself (it only takes a few minutes).

Just remember, no matter what anyone tries to tell you about this backdoor being "undetectable" by modern security tools:

The behavior caused by the `XZ` backdoor was detectable. If any of our customers had been impacted (they weren't, thankfully), they'd have known instantly—even before the vulnerability was named or disclosed.

Ready to detect and stop XZ Backdoor? Try Oligo ADR Today