XZ Backdoor: How to check if your systems are affected

By DCSO CyTec Blog

Published: 2024-04-09 · Archived: 2026-04-06 01:25:54 UTC

7 min read

Apr 8, 2024

A principal software engineer at Microsoft by the name of Andres Freund accidentally stumbled upon a backdoor within XZ, the popular compression library. This discovery happened because he noticed both a 500ms delay for ssh as well as the sshd processes using a surprising amount of CPU, despite immediately failing because of wrong usernames etc. He researched this and on March 29th, 2024 posted his findings on oss-security, a popular mailing list, avoiding a worldwide disaster in the open-source community.

Press enter or click to view image in full size

Photo by Markus Spiske on Unsplash

Blog authored by Veronika Wiederkehr and Sooi Vervloessem.

What is XZ Utils?

XZ Utils is a set of open-source tools for lossless data compression based on the Lempel-Ziv-Markov chain algorithm widely utilized across Unix-like operating systems. These include Liblzma, a general-purpose data compression library with a zlib-like API, as well as the eponymous xz. At the time of writing, GitHub shutdown the XZ repository for further investigation.

The XZ Backdoor explained

Jia Tan, believed to be a malicious contributor and maintainer of XZ Utils, worked more than two years on slowly gaining trust and implementing malicious pieces of code to the XZ Utils repository. It is currently unknown if this is their real name, or if this is even one individual and not a nation state.

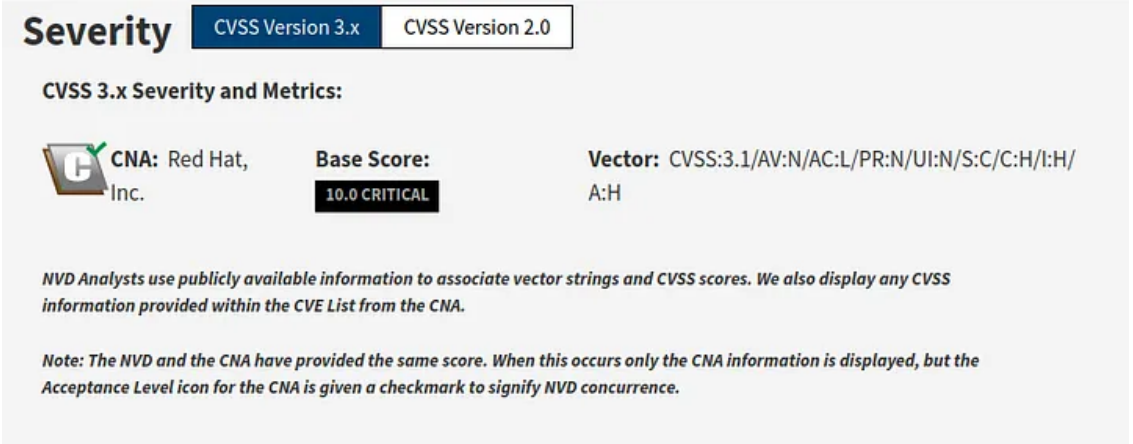
The backdoor is obfuscated so well that at the time of writing, researchers are still trying to decipher and analyze certain parts.

It was designed so that the XZ backdoor would be injected into OpenSSH binaries on Linux at several stages during compilation.

Using this backdoor, a threat actor (in possession of the private counterpart of a hard-coded ED448 public key) can remotely execute code if a target machine is running a combination of Linux, systemd and an OpenSSH server binary that has been poisoned with the backdoor, and the SSH server port is directly exposed to the internet. A more technical and detailed explanation can be found in the next chapter.

The XZ backdoor is assigned CVE-2024–3094 with a Base Score of 10.0 (Critical):

Press enter or click to view image in full size



Source: NIST

Something worth noting is that according to GitHub user thesamesam, no log messages at the INFO level or higher are written by OpenSSH when a successful exploitation occurs. This makes the detection by log analysis in production environments highly unlikely.

Workflow of the Backdoor

The m4/build-to-host.m4 macro is executed during the build process. This macro uncorrupts a malformed XZ file, bad-3-corrupt_lzma2.xz, that has been disguised as a code test. Afterwards, a malicious Bash script is extracted

and decrypted from another decoy test archive, good-large-compressed.lzma.

This Bash script will extract a binary file, decrypt it and save it as liblzma_la-crc64-fast.o. Jia Tan also commited a `.` in the CMake check for landlock support, so that this check always fails causing the landlock support to be detected as absent:

Press enter or click to view image in full size

```
diff --git a/CMakeLists.txt b/CMakeLists.txt
index 76700591059711e3a4da5b45cf58474dac4e12a7..d2b1af7ab0ab759b6805ced3dff2555e2a4b3f8e 100644 (file)
--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -901,10 +901,29 @@ endif()

 # Sandboxing: Landlock
 if(NOT SANDBOX_FOUND AND ENABLE_SANDBOX MATCHES "^ON$|^landlock$")
-    check_include_file(linux/landlock.h HAVE_LINUX_LANDLOCK_H)
+    # A compile check is done here because some systems have
+    # linux/landlock.h, but do not have the syscalls defined
+    # in order to actually use Linux Landlock.
+    check_c_source_compiles("
+        #include <linux/landlock.h>
+        #include <sys/syscall.h>
+        #include <sys/prctl.h>
+.
+        void my_sandbox(void)
+        {
+            (void)prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0);
+            (void)SYS_landlock_create_ruleset;
+            (void)SYS_landlock_restrict_self;
+            (void)LANDLOCK_CREATE_RULESET_VERSION;
+            return;
+        }
+
+        int main(void) { return 0; }
+        "
+    HAVE_LINUX_LANDLOCK)

-    if(HAVE_LINUX_LANDLOCK_H)
-        set(SANDBOX_COMPILE_DEFINITION "HAVE_LINUX_LANDLOCK_H")
+    if(HAVE_LINUX_LANDLOCK)
+        set(SANDBOX_COMPILE_DEFINITION "HAVE_LINUX_LANDLOCK")
         set(SANDBOX_FOUND ON)

         # Of our three sandbox methods, only Landlock is incompatible
```

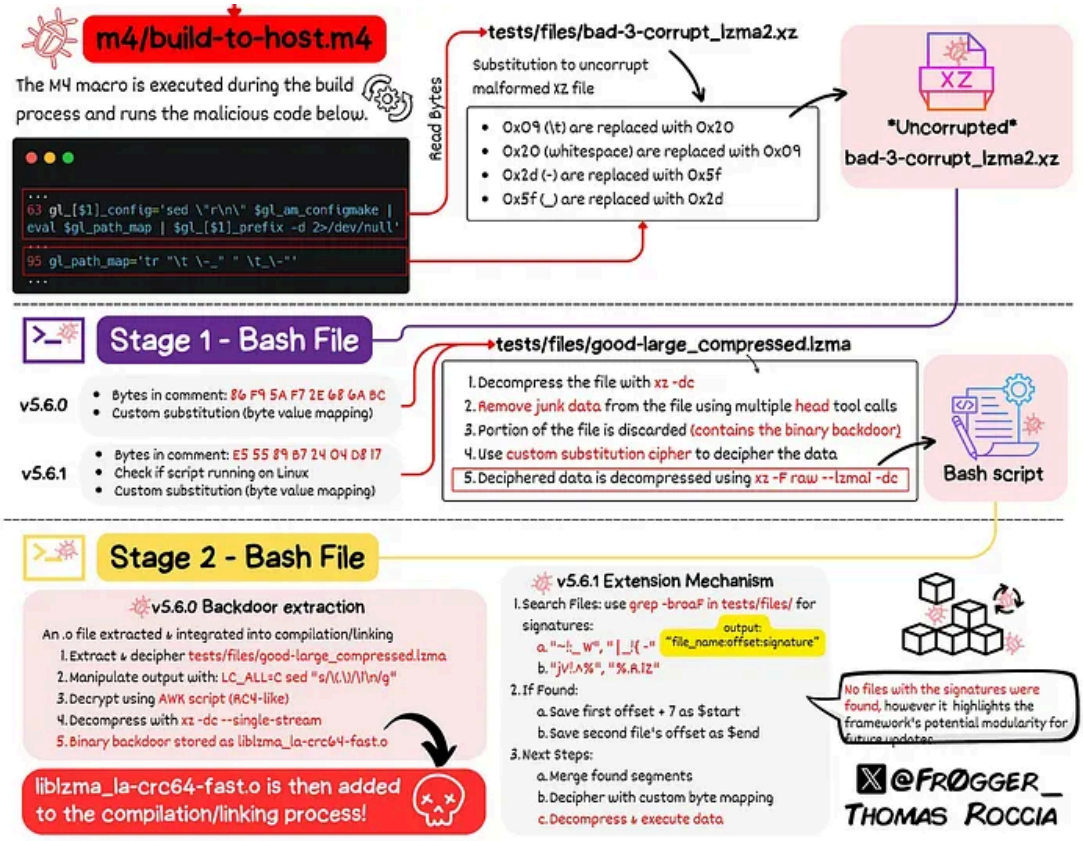
Source: Tukaani

The backdoor will hijack pre-authentication RSA decryption function calls in OpenSSH server binaries. When a connection towards the SSH server is verified by the backdoor, the adversary's payload is extracted and piped into system(). This makes the target machine vulnerable to remote code execution.

Jia Tan even added an extension mechanism, which makes the backdoor scalable and avoids the need to change the binary every time. The idea is to just add more signature files that contain malicious payloads. According to Gynvael Coldwind, the idea is to avoid modifying 'bad' and 'good' test files over and over again, since this is pretty suspicious.

A detailed yet well-structured scheme from Thomas Roccia about the workflow can be found below:

Press enter or click to view image in full size

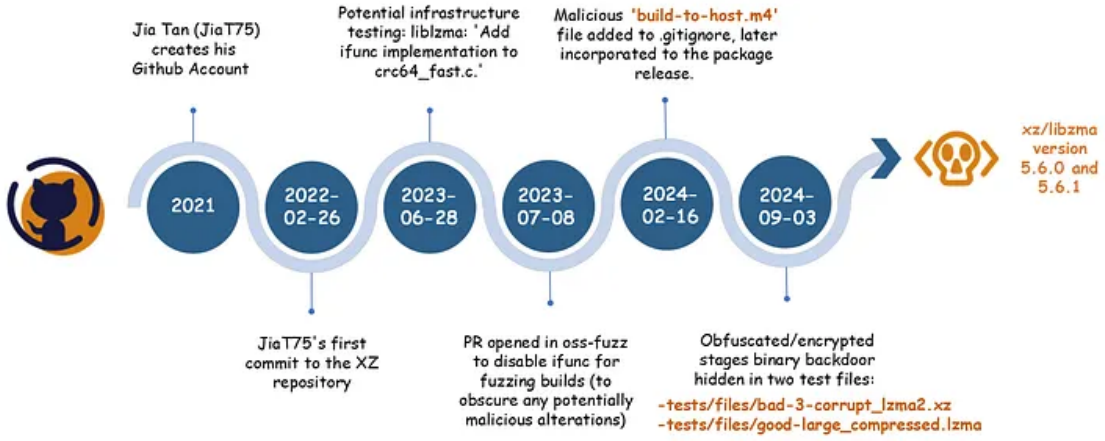


Source: [Thomas Roccia]

Timeline

To visualize the supply chain attack, underneath you can find a timeline:

Press enter or click to view image in full size



Timeline based upon scheme from [Thomas Roccia]

Affected XZ Versions

The XZ versions affected by this backdoor at the time of writing are versions 5.6.0 and 5.6.1. These versions are luckily not used in most stable versions of Linux distributions. The versions occur only in rolling-release, ie

mostly experimental versions of Linux distributions. There is a possibility that there are other backdoors Jia Tan has placed into XZ.

Affected Operating Systems

Underneath, you can find a list of affected operating systems:

- Kali Linux (only those that were available between March 26–29)
- openSUSE Tumbleweed/MicroOS (available from March 7-28)
- Fedora 41, Fedora Rawhide, and Fedora Linux 40 beta
- Debian (testing, unstable (sid) and experimental distributions only)
- Arch Linux
- Alpine Edge (active development)

Please note that additional operating systems and distributions might be added over time.

How to check if your system is affected?

In order to verify if a system is affected by the XZ backdoor, you should check the following things. These checks only aim at unveiling the **CVE-2024–3094** backdoor, so please keep in mind that there might be other backdoors that are yet to be discovered.

Check affected version
To check if any known backdoored XZ version is present on your system, execute the following command:

```
strings `which xz` | grep '5\.6\.[01]'
```

If there is no output, it means that no such binary was found in any of the $PATHs specified.

Check Liblzma library
To check whether there is an OpenSSH server loading the Liblzma library, you can execute the following command:

```
lsof -p $(ps -aux | grep 'sshd' | grep 'listener' | awk '{print $2}') | grep '\.so' | grep 'liblzma'
```

In the output you can see which version of the Liblzma library is loaded in the memory. If that version matches one of the affected versions, please take further actions.

If you get no output, we can assume that there is no SSH server loading the Liblzma library in the memory of the system.

Use the XZ Backdoor scanner
Another option is using the XZ Backdoor scanner from Binarly.

This scanner looks for the backdoor code in an ELF file which the user can upload (<5Mb).

Use the XZ Backdoor Detector
There is a detector for the CVE-2024–3094 on GitHub. It checks whether a malicious version of xz or liblzma is installed on the system.

The tool also checks whether the currently installed SSH server (sshd) links to liblzma. SSH servers that do not link to lzma are not affected by CVE-2024–3094 as the backdoor will never activate.

To quickly analyze your current system, you can execute the following commands:

```
git clone https://github.com/jfrog/cve-2024-3094-tools.git
cd cve-2024-3094-tools/cve-2024-3094-detector/
./cve-2024-3094-detector.sh
```

This is what the output could look like for a system that is not affected:

Press enter or click to view image in full size

You can find the GitHub repository here.

How to check if your container is affected?

Grype is a vulnerability scanner for container images and filesystems. It is very easy to use and is able to find vulnerabilities for major operating systems.

Installation Grype

To install Grype, please follow the instructions in the installation section of the GitHub page.

Verify the artifacts

Checksums are applied to all artifacts, and the resulting checksum file is signed using cosign.

You need the following tool to verify signature:

- Cosign

Verification steps are as follow:

1. Download the files you want, and the checksums.txt, checksums.txt.pem and checksums.txt.sig files from the releases page
2. Verify the signature:

```
cosign verify-blob <path to checksum.txt> \
--certificate <path to checksums.txt.pem> \
--signature <path to checksums.txt.sig> \
--certificate-identity-regexp 'https://github\.com/anchore/grype/\.github/workflows/.+' \
--certificate-oidc-issuer "https://token.actions.githubusercontent.com"
```

Once the signature is confirmed as valid, you can proceed to validate that the SHA256 sums align with the downloaded artifact:

```
sha256sum --ignore-missing -c checksums.txt
```

Getting started

Install the binary, and make sure that `grype` is available in your path. To scan for vulnerabilities in an image:

```
grype <image>
```

The above command scans for vulnerabilities that are visible in the container (i.e., the squashed representation of the image). To include software from all image layers in the vulnerability scan, regardless of its presence in the final image, provide `--scope all-layers`:

```
grype <image> --scope all-layers
```

To run grype from a Docker container so it can scan a running container, use the following command:

```
docker run --rm \
--volume /var/run/docker.sock:/var/run/docker.sock \
```

```
--name Grype anchore/grype:latest \
$(ImageName):$(ImageTag)
```

You can specify a lot of options and even scan on specific vulnerabilities. For more information, see the GitHub repository.

Exploitation in the wild

Currently, there are no known/public cases in which the backdoor was leveraged. If you are affected by this backdoor and have the suspicion that there were post-exploitation activities, we would love to hear from you.

Conclusion

There are multiple ways to check if your system has been affected by the XZ backdoor. The XZ Backdoor Scanner and XZ Backdoor Detector offer a fast way to check your systems. For containers, there is a vulnerability scanner called ‘Grype’. Up until the time of writing, there has been no known case in which the backdoor was exploited.
