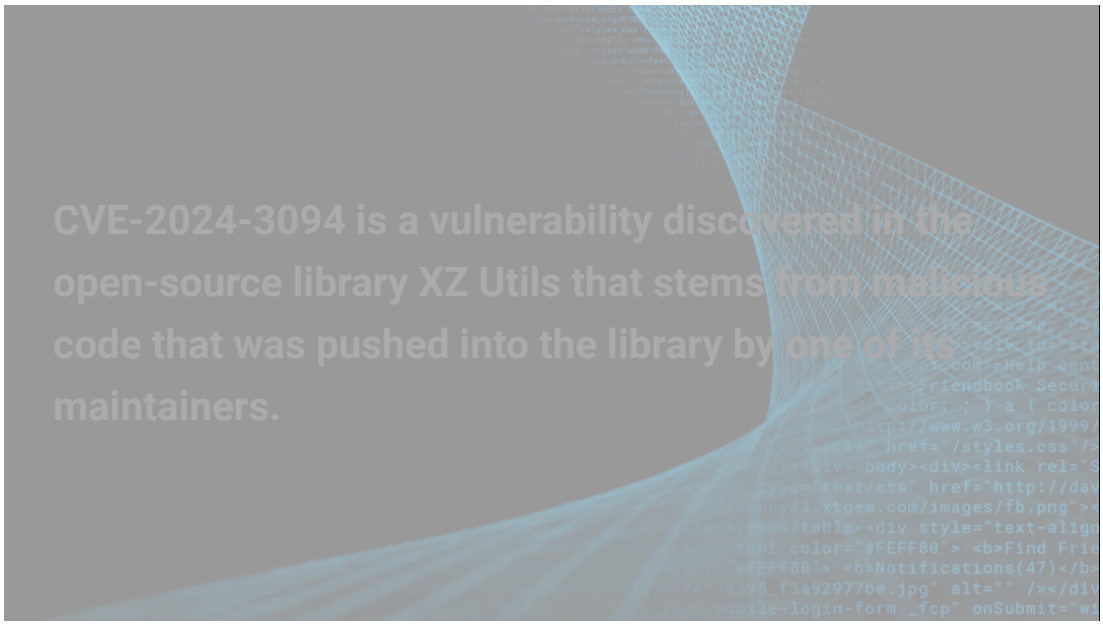
XZ Utils Backdoor — Everything You Need to Know, and What You Can Do

Akamai Security Intelligence Group

April 01, 2024



Executive summary

- CVE-2024-3094 is a vulnerability discovered in the open-source library XZ Utils that stems from malicious code that was pushed into the library by one of its maintainers.
- It was originally reported as an SSH authentication bypass backdoor, but further analysis indicates that the backdoor actually enables remote code execution (RCE).
- The threat actor started contributing to the XZ project almost two years ago, slowly building credibility until they were given maintainer responsibilities. Such long-term operations are usually the realm of state-sponsored threat actors, but specific attribution does not currently exist.
- Since the backdoor affects the latest XZ Utils releases, the recommended course of action is to downgrade to an uncompromised release. In this blog post, we offer other

potential mitigations to limit the blast radius of the attack.

Jump to action items

Backstory

XZ Utils,, and its underlying library liblzma, are open-source projects that implement the lzma compression and decompression. They are included in many Linux distributions out of the box, are very popular with developers, and are used extensively throughout the Linux ecosystem.

Almost two years ago, a developer under the name of Jia Tan joined the project and started opening pull requests for various bug fixes or improvements. So far, nothing is out of the ordinary; this is how things work in the open-source world. Eventually, after building trust and credibility, Jia Tan began to receive permissions for the repository — first, commit permissions and, eventually, release manager rights.

It seems that as part of the effort to gain these permissions, Jia Tan used an interesting form of social engineering: They used fake accounts to send myriad feature requests and complaints about bugs to pressure the original maintainer, eventually causing the need to add another maintainer to the repository.

After contributing to the code for approximately two years, in 2023 Jia Tan introduced a few changes to XZ that were included as part of release 5.6.0. Among these changes was a sophisticated backdoor.

The backdoor

The backdoor is quite complex. For starters, you won't find it in the xz GitHub repository (which is currently disabled, but that's besides the point). In what seems like an attempt to avoid detection, instead of pushing parts of the backdoor to the public git repository, the malicious maintainer only included it in source code tarball releases. This caused parts of the backdoor to remain relatively hidden, while still being used during the build process of dependent projects.

The backdoor is composed of many parts introduced over multiple commits:

- Using IFUNCs in the build process, which will be used to hijack the symbol resolve functions by the malware
- Including an obfuscated shared object hidden in test files
- Running a script set during the build process of the library that extracts the shared object (not included in the repository, only in releases, but added to .gitignore)
- Disabling landlocking, which is a security feature to restrict process privileges

The execution chain also consists of multiple stages:

- The malicious script *build-to-host.m4* is run during the library's build process and decodes the "test" file *bad-3-corrupt_lzma2.xz* into a bash script
- The bash script then performs a more complicated decode process on another "test" file, *good-large_compressed.lzma*, decoding it into another script

- That script then extracts a shared object *liblzma_la-crc64-fast.o*, which is added to the compilation process of liblzma

This process is admittedly hard to follow. We recommend Thomas Roccia's infographic for a great visual reference and in-depth analysis.

The shared object itself is compiled into liblzma, and replaces the regular function name resolution process. During (any) process loading, function names are resolved into actual pointers to the process memory, pointing at the binary code. The malicious library interferes with the function resolving process, so it could replace the function pointer for the OpenSSH function RSA_public_decrypt (Figure 1).

It then points that function to a malicious one of its own, which according to research published by Filippo Valsorda, extracts a command from the authenticating client's certificate (after verifying that it is the threat actor) and passes it on to the system() function for execution, thereby achieving RCE prior to authentication.

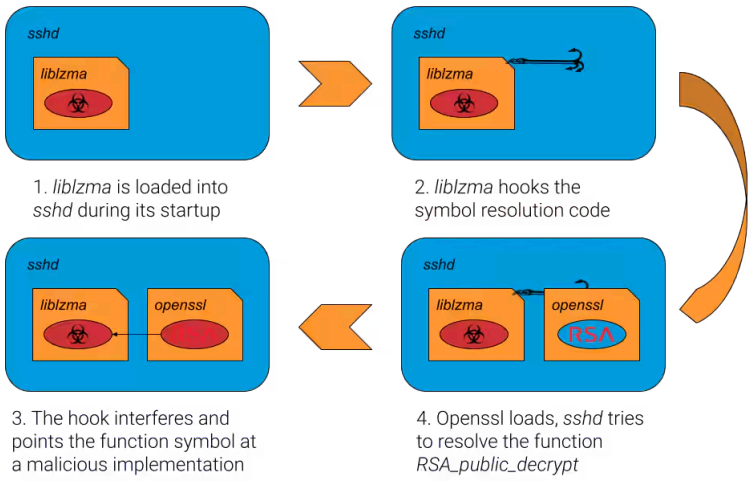


Fig. 1: The liblzma hooking process

For a more detailed explanation of the backdoor parts, you can read Andres Freund's post on openwall.

Potential impact

Currently, it appears as though the backdoor is added to the SSH daemon on the vulnerable machine, enabling a remote attacker to execute arbitrary code. This means that any machine with the vulnerable package that exposes SSH to the internet is potentially vulnerable.

This backdoor almost became one of the most significant intrusion enablers ever — one that would've dwarfed the SolarWinds backdoor. The attackers were almost able to gain immediate access to any Linux machine running an infected distro, which includes Fedora, Ubuntu, and Debian. Almost.

There was only one thing that stopped that from happening — Andres Freund. After investigating a 500 ms latency issue that was introduced after a software update, Andres was able to trace the issue back to the xz package and ultimately identify the backdoor.

This obviously raises a lot of concerns. We got lucky. If this backdoor was not detected by a curious engineer, how long would it have remained active?

And perhaps even more concerning: What if this has happened before?

Detection and mitigation

Version control

The Cybersecurity and Infrastructure Security Agency (CISA) recommended course of action is to downgrade to an uncompromised version, such as 5.4.6.

To know which version of XZ Utils or liblzma you currently have on your systems, you can run the following query in Akamai Guardicore Segmentation Insight that will look for loaded instances of the liblzma library (Figure 2).

Copy

```
SELECT DISTINCT path AS liblzma_path
FROM process_memory_map
WHERE LOWER(path) LIKE "%liblzma%"
```

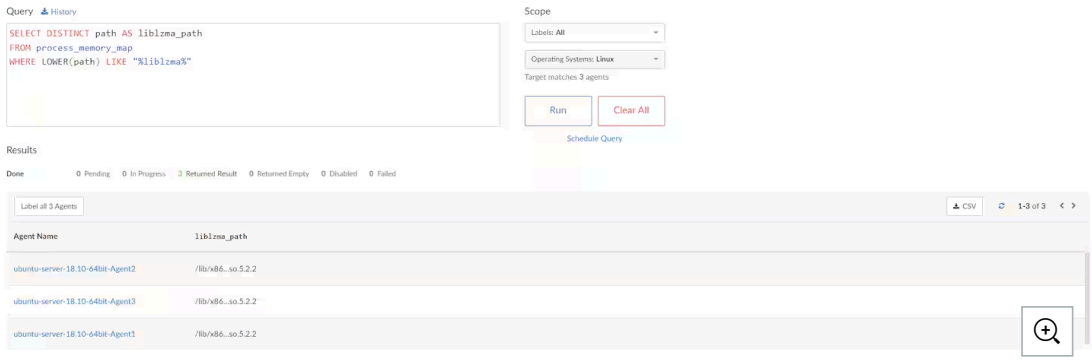


Fig. 2: Querying for loaded instances of liblzma

Alternatively, you can run the following query to find the package manager for the installed version.

Copy

```
SELECT name AS vulnerable_item, 'DEB' AS type, version
FROM deb_packages
WHERE (LOWER(name) LIKE '%xz-utils%' OR LOWER(name) LIKE '%liblzma%')

UNION

SELECT name AS vulnerable_item, 'RPM' AS type, version
FROM rpm_packages
WHERE (LOWER(name) LIKE '%xz-utils%' OR LOWER(name) LIKE '%liblzma%')
```

Of course, you can also filter to show only vulnerable assets.

Copy

```
SELECT path AS vulnerable_item, "Loaded Library" AS type, '5.6%' AS version
FROM process_memory_map
```

```
WHERE LOWER(path) LIKE "%liblzma%5.6%"
```

Copy

```
SELECT name AS vulnerable_item, 'DEB' AS type, version
FROM deb_packages
WHERE (LOWER(name) LIKE '%xz-utils%' OR LOWER(name) LIKE '%liblzma%')
AND version LIKE '5.6.%'

UNION

SELECT name AS vulnerable_item, 'RPM' AS type, version
FROM rpm_packages
WHERE (LOWER(name) LIKE '%xz-utils%' OR LOWER(name) LIKE '%liblzma%')
AND version LIKE '5.6.%'
```

Threat hunting

Since the backdoor actually executes system commands, and isn't just allowing authentication, it might be possible to detect this behavior via process tracking.

Usually, during logon, a new shell is created for the logging user, and runs the default shell process (like bash). However, with this backdoor, the malicious command is actually executed by the SSH daemon process, *sshd*, which could trigger an anomaly.

Our threat hunting service, Akamai Hunt, has methods in place to detect such anomalies; for example, by constantly tracking a baseline of process activity and their child processes.

Kill switch

According to some analyses of the backdoor, it appears to have an environment variable kill switch. Adding the key *yolAbejyiejuvnup=Evjtgvsh5okmkAvj* to the system's environment variables may disable the backdoor.

References

- Backdoor in upstream xz/liblzma leading to ssh server compromise
- FAQ on the xz-utils backdoor
- Filippo Valsorda on X
- CISA advisory
- RedHat CVE

See more research