liblzma - OSS and Backdoors

Exploring the xz-utils Backdoor, its Emergence and its Impact on FOSS and OSS

9525469
xnacly.me
Applied Computer Science
DHBW Mosbach

June 25, 2024

Abstract

In recent years, several vulnerabilities in the open-source software supply chain were discovered. The most recent being the intentionally placed backdoor in the compression library named *liblzma*. This paper aims to explore the implementation of said backdoor while highlighting the insertion of the backdoor and the inserters use of social engineering enabling their placement in the leadership of the project. Furthermore ways of preventing similar attacks are presented and evaluated on the example of the *liblzma* situation.

1 Introduction

FOSS[1] is generally defined as software the user can *"[...] run, copy, distribute, study, change and improve [...]"* [2]. This requires the source to be available and enables the dependence of other software on subsets or the entirety of the code. On the other hand, source available or OSS[2] are distinct from FOSS software. Some licenses do not require the resulting product to be licensed under the same license as its dependencies, such as the MIT license[3]. It therefore differs from the GPL[4] and software licensed with the MIT-Licence can therefore not be referred to as free open-source software, but rather as open-source software.

Most OSS-projects accept contributions from individuals and enterprises. This is wanted and required to support the actuality of said software. Most OSS projects accept changes matching their pre-defined contribution guidelines and credit the contributor for their addition. These contributors often use the software they are contributing to and therefore make changes they care for, such as adding drivers for new devices to the Linux kernel [5].

However, other independent OSS contributors are abusing the contribution system by exploiting the trust the unpaid maintainers have in the quality of the submitted changes. Specifically, this refers to manipulating maintainers and inserting oneself into the group of by applying pressure on said group. As was the case with `liblzma` or the `xz-utils` OSS library.

[1] Free and Open-Source Software [1]

[2] Open-Source Software

[3] Requires the license to be present in *"all copies or substantial portions of the Software"* [3]

[4] Requires all copies of the software to be licensed as GPL [4]

1.1 Dependence on FOSS and OSS

Open source software is often divided into reusable components, such as libraries or toolkits implementing a specific feature, and built upon by other software. The goal is to use tried and tested components in the creation of new OSS, thus building on field tested and established software found in the OSS community.

Not only does OSS depend on other libraries from the OSS ecosystem. Proprietary software also makes use of said OSS components, while being forced to adhere to their terms, as declared in their respective licenses [6].

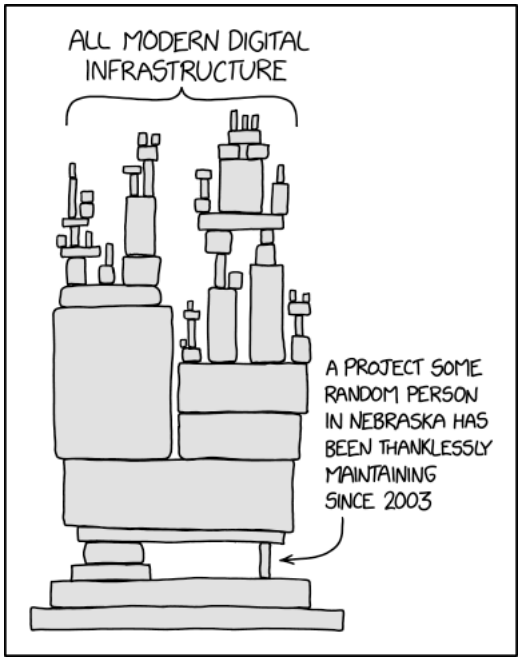

Figure 1: Dependency [7]

Commonly used examples for said libraries are `libcurl`, which provides multi protocol file transfers [8], `raylib` which is a library for video games programming [9] and the `sqlite` library, that implements a in process database [10]. These library examples are so widely used, a vulnerability in them would impact the security of the whole software space.

1.2 Supply Chain Security

Supply chain security is referring to the fact of ensuring the dependencies of a given software are to be considered secure and making sure this property can be accessed to be true. This can be achieved by keeping the development toolchain used for creating said software up to date, thus patching and removing found vulnerabilities [11]. Other ways of establishing the security of a dependency is to manually evaluate the source code of the given dependency.

However most largely used open source software is thoroughly tested and famously so. `sqlite` prides itself as being the *"most used and deployed database engine"* [12] and *"[...] the project has 590 times as much test code and test scripts [as lines of source code]"* [13].

Considering the large amount of memory safety errors[5], often caused by invalid or maliciously crafted input, projects entirely focussed around detecting said vulnerabilities, such as `OSS-Fuzz` [15], saw their inception.

1.3 xz-utils and liblzma

`xz-utils` refers to a c implementation of the `xz` compression algorithm and format using the Lempel–Ziv–Markov-Chain. It is written to comply with the C99 standard and consists of several components. One of these components is, as previously introduced, a library providing an API for compression and decompression, named `liblzma` [16]. According to the components documentation, its API is based on the `lzma` SDK but includes heavy modifications necessary for the `xz-utils` suite [17].

[5] 70% according to [14]

2 Backdoor Exploration

According to [18], a backdoor refers to a hidden method of gaining entry to a system bypassing security measures, such as biometric or password based authentication. They can be implemented in cryptographic algorithms, on the hardware level or in an application. Backdoors can be used to remotely access systems and are often hidden inside commonly used non-malicious software.

2.1 Implementation

`CVE-2024-3094` was assigned to the `libzlma` backdoor with the highest possible CVSS Score: 10.0. This assessement was made due to the severness of the included remote code execution [19].

2.2 Social Engineering & Pressure on OSS Maintainer

- mental health issues
- attacker helps maintainer
- attacker gains maintainer status
- attacker manages project home page and oss-fuzz email
- attacker announces and publishes releases

2.3 Build Pipeline Manipulation

- attacker makes changes to test files
- attacker makes changes to build pipeline (autoconf)
- attacker injects malicious shell code into test files
- test files are being decompressed, decrypted and injected into the shared object

2.4 IFUNC & CPU specific Features

- IFUNC is used to switch to a hardware supported function implementation at runtime
- abused by the attacker to replace the `openssh` function `RSA_public_decrypt()`
- replaced with execution of injected shell code via `system()`

2.5 Indirect Dependence on libzlma

- `openssh` does not depend on `libzlma`
- common patches link `openssh` dynamically against `libsystemd`, which links agains `liblzma`
- making `openssh` vulnerable via indirect dependence

2.6 Modifying ELF & Executing Shell Code

- `openssh` does not depend on `libzlma`
- common patches link `openssh` dynamically against `libsystemd`, which links agains `liblzma`
- making `openssh` vulnerable via indirect dependence

2.7 Prerequisites for the Backdoor

- built with `GCC` and `glibc`
- shared object is opened on `x86-64`
- built by `dpkg` or `rpm`

3 Response

3.1 Detection

- Detected by Andres Freund on 29th March 2024
- PostgreSQL developer
- noticed high CPU usage and Valgrind errors when attempting to connect via SSH
- compromised version not yet deployed to production systems but dev builds

3.2 Software Distributors Reactions

- Red Hat, SUSE and Debian downgraded `libzlma` to its previous version
- Ubuntu held the beta release of Ubutun 24.04 LTS back by a week and rebuild all binaries of the distros packages

3.3 Supply Chain Security

- should critical parts of the software supply chain depend on unpaid volunteers
- keeping the supply chain secure

4 Prevention

4.1 Funding FOSS and OSS

4.2 Vetting Dependency

4.3 Appreciation for FOSS Maintainers

References

[1] R. M. S. (RMS). (2021), [Online]. Available: `https://www.gnu.org/philosophy/pragmatic.html` (visited on 06/04/2024).

[2] F. S. Foundation. (2024), [Online]. Available: `https://www.gnu.org/philosophy/free-sw.html` (visited on 06/04/2024).

[3] Opensource.org. (2024), [Online]. Available: `https://opensource.org/license/MIT` (visited on 06/04/2024).

[4] Opensource.org. (2024), [Online]. Available: `https://opensource.org/license/gpl` (visited on 06/04/2024).

[5] T. kernel development community. “Linux - introduction - device drivers.” (2021), [Online]. Available: `https://linux-kernel-labs.github.io/refs/heads/master/lectures/intro.html#device-drivers` (visited on 06/10/2024).

[6] D. Stenberg. “Companies using curl in commercial environments.” (2024), [Online]. Available: `https://curl.se/docs/companies.html` (visited on 06/09/2024).

[7] xkcd. “Dependency.” (), [Online]. Available: `https://xkcd.com/2347/` (visited on 06/09/2024).

[8] D. Stenberg. “Libcurl - the multi-protocol file transfer library.” (2024), [Online]. Available: https://curl.se/libcurl/ (visited on 06/09/2024).

[9] raysan5. “Raylib is a simple and easy-to-use library to enjoy videogames programming.” (2024), [Online]. Available: https://www.raylib.com/ (visited on 06/09/2024).

[10] D. R. Hipp. “What is sqlite?” (2024), [Online]. Available: https://www.sqlite.org/index.html (visited on 06/09/2024).

[11] M. Stapelberg. “Supply chain security with go.” (2024), [Online]. Available: https://media.ccc.de/v/gpn22-438-supply-chain-security-with-go (visited on 06/07/2024).

[12] J. M. Dwayne Richard Hipp Dan Kennedy. “Most widely deployed and used database engine.” (2022), [Online]. Available: https://www.sqlite.org/mostdeployed.html (visited on 06/20/2024).

[13] J. M. Dwayne Richard Hipp Dan Kennedy. “How sqlite is tested.” (2022), [Online]. Available: https://www.sqlite.org/testing.html (visited on 06/20/2024).

[14] T. C. Project. “Memory safety.” (), [Online]. Available: https://www.chromium.org/Home/chromium-security/memory-safety/ (visited on 06/10/2024).

[15] G. O. Source. “Oss-fuzz.” (2024), [Online]. Available: https://google.github.io/oss-fuzz/ (visited on 06/20/2024).

[16] L. Collin. “Xz utils.” (2024), [Online]. Available: https://tukaani.org/xz/#_introduction (visited on 06/21/2024).

[17] cy. “Common code <> different backdoors.” (2024), [Online]. Available: https://media.ccc.de/v/gpn22-304-common-code-different-backdoors (visited on 06/05/2024).

[18] C. Wysopal and C. Eng. “Static detection of application backdoors.” (2007), [Online]. Available: https://www.veracode.com/sites/default/files/Resources/Whitepapers/static-detection-of-backdoors-1.0.pdf (visited on 06/24/2024).

[19] R. Hat. “Cve-2024-3094.” (2024), [Online]. Available: https://access.redhat.com/security/cve/CVE-2024-3094 (visited on 06/24/2024).

Appendix