CVE-2022-37434
zlib inflate function causes out-of-bounds write

109550060 陳星宇

What is CVE-2022-37434?

- zlib through 1.2.12 has a heap-based buffer over-read or buffer overflow in inflate in inflate.c via a large gzip header extra field.
- only applications that call inflateGetHeader are affected.

How is the CVE discovered?

- An issue in Github with curl tag turned out a zlib CVE issue.
- Test224 in Curl test suite send a HTTP request to a server with content that wants to be encoded accompanied with large extra header field. It should be ignored and refuse to compress the content if the extra header field exceeds max length. However, the developer didn’t do the error check for the header length. This will result in heap overflow and undesired behavior of writing since the compressed content will be “memcpied” into a out-of-range memory.

CVSS scores and scoring vectors

Severity level is medium in Red Hat and severe in NVD.

As Red Hat stated that zlib functions are bundled in their products, it is unable to be utilized by the attacker, which leads to low confidentiality and integrity risks.

CVSS v3 Score Breakdown

	Red Hat	NVD
CVSS v3 Base Score	7.0	9.8
Attack Vector	Network	Network
Attack Complexity	High	Low
Privileges Required	None	None
User Interaction	None	None
Scope	Unchanged	Unchanged
Confidentiality Impact	Low	High
Integrity Impact	Low	High
Availability Impact	High	High

Reproduce the CVE environment

Since only the V1.2.2 patched zlib is vulnerable with the attack, we have to prepare a server with v1.2.2 zlib first, then prepare the text file that will possibly lead to the exploit.

Reproduce the exploitation

- Prepare two cpp files, “inflate.c” and “deflate.c”. “deflate.c” is used to compress the data and “inflate.c” is used to decompress the data. Use deflateSetHeader() to set gzip header and then use inflateGetHeader() to trigger the exploit.

Analysis

- It is a trivial bug to an individual PC user, as ordinary folk only uses zlib’s inflate() and deflate() function, while this bug is only triggered by using inflateGetHeader() function as it retrieves the extra content field by checking its extra_len field. However, for those HTTP server who use gzip as content-encoding method, it is definitely a critical issue.

Explanation

- To reproduce the experiment, you can use make to start and make clean to recover to its original status.
- Use deflateInit2 to specify gzip encoding with header(by setting windowsbit to 15 | 16) in the deflate.c.
- Use deflateSetHeader to fill in the information of the header and fill in the extra_len and extra_field, with extra_len<len(extra_field).
- Use inflateInit2 to specify gzip decoding with header(by setting windowsbit to 15 | 16) in the inflate.c.
- Use inflateGetHeader to retrieve the header information, and it will trigger the buggy code in inflate.c in zlib 1.2.12 version.

Result of the experiment

- As there’s no public exploit on the Internet, I tried to reproduce it by myself. However, when I finished all the part in my topology, I found it not working when I call the InflateGetHeader() function. Although it output that it was successful, it did not resolve the header correctly. Therefore, it successfully decompressed the compressed data back and did not trigger the buggy code.