How image-based heap overflows work

An image-based heap overflow is a memory-safety bug where a program that decodes an image writes past the end of a heap buffer it allocated for that image. It happens because the decoder sizes the buffer from values it reads out of the file (width, height, chunk lengths), and an attacker sets those values so the buffer comes out too small while the copy that fills it uses the real, larger amount of data. The out-of-bounds write corrupts nearby heap memory, and with careful heap layout it can be turned into remote code execution, often with no action beyond viewing or receiving the image.

HOW IT WORKS

01 How the size calculation goes wrong

The bug almost always lives in the arithmetic that sizes the buffer. A decoder typically computes the buffer size from fields in the file, for example `width * height * bytes_per_pixel`, or the length of a compressed data chunk.

An attacker controls those fields, so they can pick values that make the multiplication wrap around. On a 32-bit size, `width * height * 4` with large enough width and height exceeds the maximum integer and wraps to a small number. The decoder allocates that small buffer, then the decode loop writes the full, real number of pixels far past the end. The same happens when a length field lies about how big a chunk is, or when a bounds check compares against the wrong variable.

> THE INTEGER-OVERFLOW-TO-HEAP-OVERFLOW PATTERN
>
> *Most image heap overflows start as an integer bug (MITRE CWE-190). The size math overflows or is off by a factor, the allocation is too small, and the copy that follows is the out-of-bounds write (CWE-787). Auditing the size calculation, not the copy, is where these are found.*

02 From a stray write to code execution

An out-of-bounds write on its own corrupts memory, which usually just crashes the program. Turning it into control is about what sits next to the overflowed buffer.

Attackers use heap grooming: they make the program allocate and free objects in a pattern that places a useful target right after the buffer they will overflow. Good targets are values the program will later trust and act on, such as a function

SOURCES

[1] MITRE CWE-122: Heap-based Buffer Overflow

[2] MITRE CWE-787: Out-of-bounds Write

[3] MITRE CWE-190: Integer Overflow or Wraparound

[4] NVD: CVE-2023-4863 (WebP heap buffer overflow)

pointer, a C++ vtable pointer, a length field, or the allocator's own metadata. Overwriting one of those redirects execution to code or data the attacker controls. Because images are often decoded before anything is shown, this can run with no clicks.

03 Why image decoders are a favourite target

Images are decoded almost everywhere: browsers, messaging apps, email clients, servers that generate thumbnails, and platforms that re-encode uploads. Much of that decoding runs automatically, before a person interacts with the content, which is what makes these bugs zero-click.

The decoders are usually written in C or C++ for speed, and the file formats are complex, with many chunk types, colour models, and optional features. That combination, wide reach plus intricate parsing in a memory-unsafe language, is why heap overflows in image libraries keep surfacing. A 2023 flaw in the WebP decoding library, tracked as CVE-2023-4863-, reached browsers and any application that bundled the same library, and was exploited in the wild.

04 How testers find them

In an assessment these are found by fuzzing the decoder: feeding it a large number of malformed and mutated image files and watching for crashes. Coverage-guided fuzzers reach deep code paths, and a memory sanitiser (ASan) turns a silent out-of-bounds write into an immediate, precise report of where it happened.

Manual review focuses on the size arithmetic and the copy loops: every place a length or dimension from the file feeds an allocation or a `memcpy`. Any multiplication or addition on attacker-controlled sizes that is not done with checked arithmetic is a candidate. Differential testing, decoding the same file with two implementations, surfaces parsers that disagree about how large the output should be.

Test the software that parses your files, before an attacker does.

securelayer7.net/learn/binary-exploitation/image-based-heap-overflows

Open online