High Performance Images —
Secure Image Delivery

Colin Bendell, Yoav Weiss, Tim Kadlec, Guy Podjarny, Nick Doyle & Mike McCall

Secure Image Delivery

The security of transforming and transformed images is an important and oft-overlooked aspect of delivery — yet it is equally as important to your brand as how you deliver images to users. What if your images were tampered with? How could your brand be tarnished if a nefarious agent accessed them?.

Secure Transport of Images

TUp until recently, the majority of the web has been delivered unencrypted. As we have all experienced, there are many locations where content can be hijacked in an unencrypted flow. Public Wi-Fi does this intentionally to force you through a captive portal before granting you access to the Internet. ISPs, with good intentions, have notoriously applied higher compression, distorting the visual quality of your brand.

Using Cache-Control: no-transform works for some, but not all, well-behaved image transformations (see Figure 13-21). But there are also not-so-well-intentioned transparent proxies that hijack image requests and replace the content with different advertisements or placeholders.

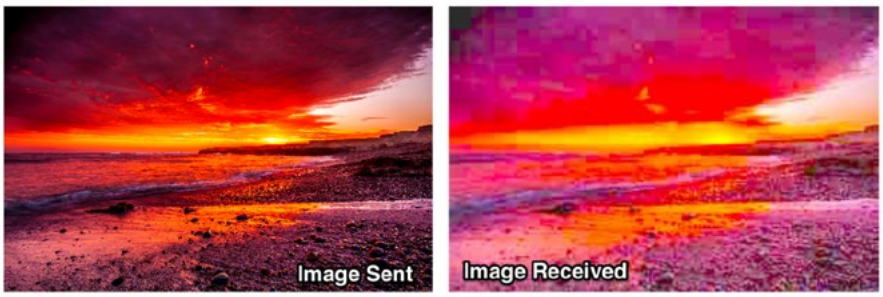

Figure 13-21. Use Cache-Control: no-transform to prevent degraded quality by ISP proxies

Securing the transport for images is straightforward. Using TLS you can ensure that the communication from user to server is trusted and that there aren't any middle boxes interfering with or mutating your content. Moving to HTTP/2 also requires the use of TLS.

Be Careful of Content Hijacking on Untrusted Wi-Fi

There have been increasing reports of free Wi-Fi hotspots found at hotels, coffee shops, and restaurants replacing web content with alternate advertising. Putting the ethical argument aside–whether service providers can generate ad revenue from offering free Wi-Fi–there are branding implications for your own web content. Hijacking content like this is only possible with unencrypted pages and images. Moving to TLS prevents man-in-the-middle interception.

Secure Transformation of Images

Securing image delivery is about more than just the transport layer. We should also be concerned about the attack surface of our transformation engines. Whether you are using an on-premise image transformation engine or an off-premise one, there are many possible vulnerabilities. Third-party and open-source libraries are extremely useful but also can introduce risk to the enterprise if not properly isolated.

An index of Common Vulnerabilities and Exposures (CVE) is maintained by Mitre (see Figure 13-22). It is critical to keep up to date with the latest known exploits on the libraries and tools used in your image transformation workflow. Isolating and patching should be part of your regular team practices.

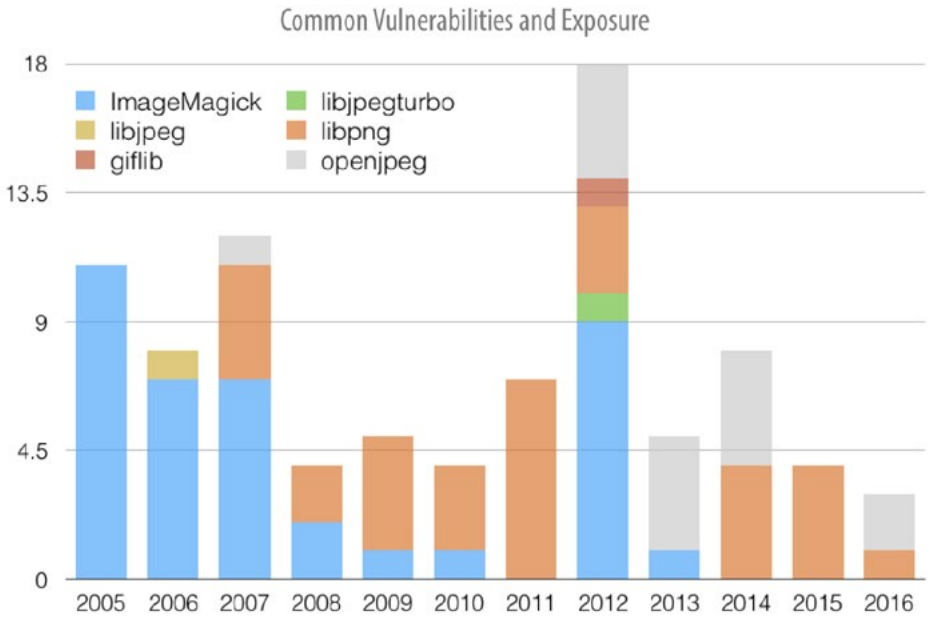

Figure 13-22. CVEs reported for ImageMagick and common image libraries

Image Exploits and Attack Vectors Common Across All Image Formats

News and blog posts of image exploits are commonplace nowadays. Most image-related attacks leverage a technique called steganography, which involves hiding a message or exploit code inside the image. No image format is exempt. In 2013, a security researcher found a backdoor that hid data within Exif headers in JPEG, and Trend Micro blogged about another, similar attack vector with JPEG. Similar attacks have used BMPs and PNGs to accomplish their malicious activities. Not only are image libraries vulnerable, image transformation engines are as well. In the spring of 2016, ImageMagick made headlines with the remote-execution vulnerability colloquially called "imagetragic".

The main concern for image-transformation engines is if a contaminated image enters for processing and, through the decode or mutation process, exploits a vulnerability. This could leverage a byte alteration from the logic edge case, checksum collision, or remote code execution. Consider that the famous Jailbreakme exploit that allowed jailbreaking on iOS 3 used a flaw in the TIFF decoder in iOS. This single flaw allowed the rooting of the entire operating system. Imagine the potential impact on your images. This vulnerability could impact subsequent images, possibly tagging them with brand-damaging messages. Just because the bytes of the image have left the processor doesn't mean there isn't residual code running on the thread. The last thing you want is all of your product images graffitied with "EAT BROCCOLI" without your realizing it (see Figure 13-23).

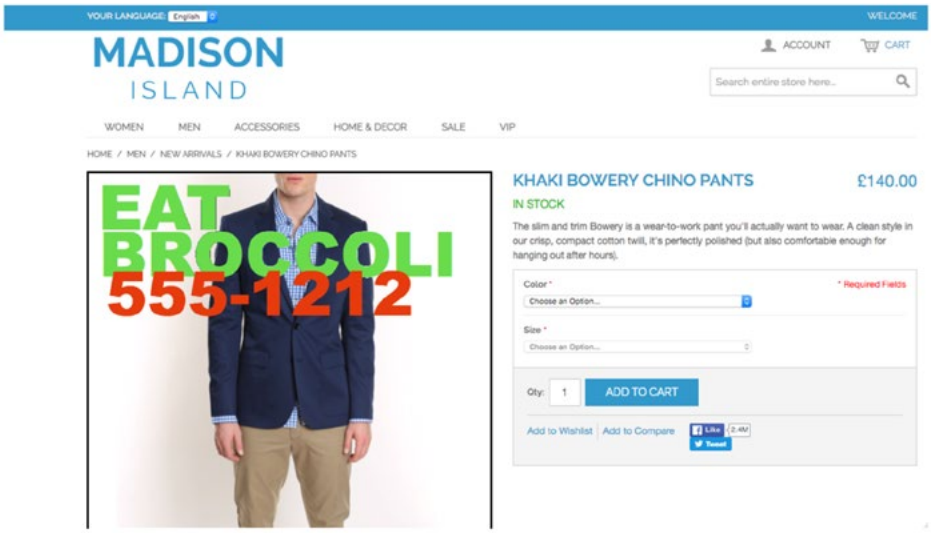

Figure 13-23. We want to avoid one image affecting other images on the platform

How could a contaminated image enter your workflow?

- User-generated images, compromised at source
- Vendor-supplied product images, compromised at source
- In-house photography, compromised by malware on the artist's laptop

It is easy to imagine how a compromised image could enter your workflow. So how can you ensure that a compromised image doesn't impact your ecosystem? How do you isolate the impact to just that compromised image? How can you minimize risk and exposure to your image transformation service?

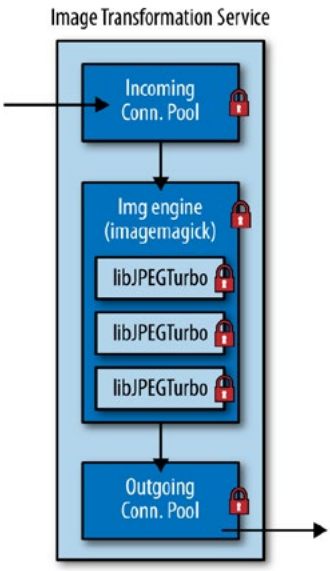

Figure 13-24. A model for secure image-transformation architecture

Secure Transformation: Architecture

Whether your image transformation is on premise or with a cloud-based SaaS provider, you should evaluate the architectural security of the transformation engine. Ideally, there should be isolation at every level of processing. You want to ensure that no single compromised image can affect other parallel threads/processes/systems that are also transforming other images. You also need to ensure there isn't any residual code that may impact the next image processed by this specific thread.

A well-secured transformation architecture should consider three major areas for isolation (see Figure 13-24):

- TCP connection pools (retrieving and storing)
- Transformation engine (e.g., ImageMagick)
- Encoding and decoding shared objects

For example:

- We need to ensure there is no way that images being sent or received via TCP (or disk) can impact another thread or process. The initiating worker should only have access to the stream of bytes for this job
- The transformation engine, such as ImageMagick, must not be able to store, execute, or preserve any state between image processing. The worker threads must be each isolated to exclusive scratch areas and restricted to access only certain system libraries. For example, the transformation engine should not be able to open up new TCP sockets or leave temporary files or memory state between jobs
- The various encoding and decoding shared objects (e.g., libjpeg-turbo) also need to be isolated. Memory state should not be allowed to persist or be accessed by parallel threads or other jobs

This is not an exhaustive list of ways to isolate and segment the architecture. Your local security team should be able to help you ensure there is no way that a maliciously tampered image can have an ecological impact on the rest of your valuable assets. If you are using a cloud solution, you should also ensure the same level of scrutiny can be applied.

The above is an excerpt from High Performance Images, developed by O'Reilly Media. For more information or to download the full book free of charge, *click here*. For more information on web images, *click here*.