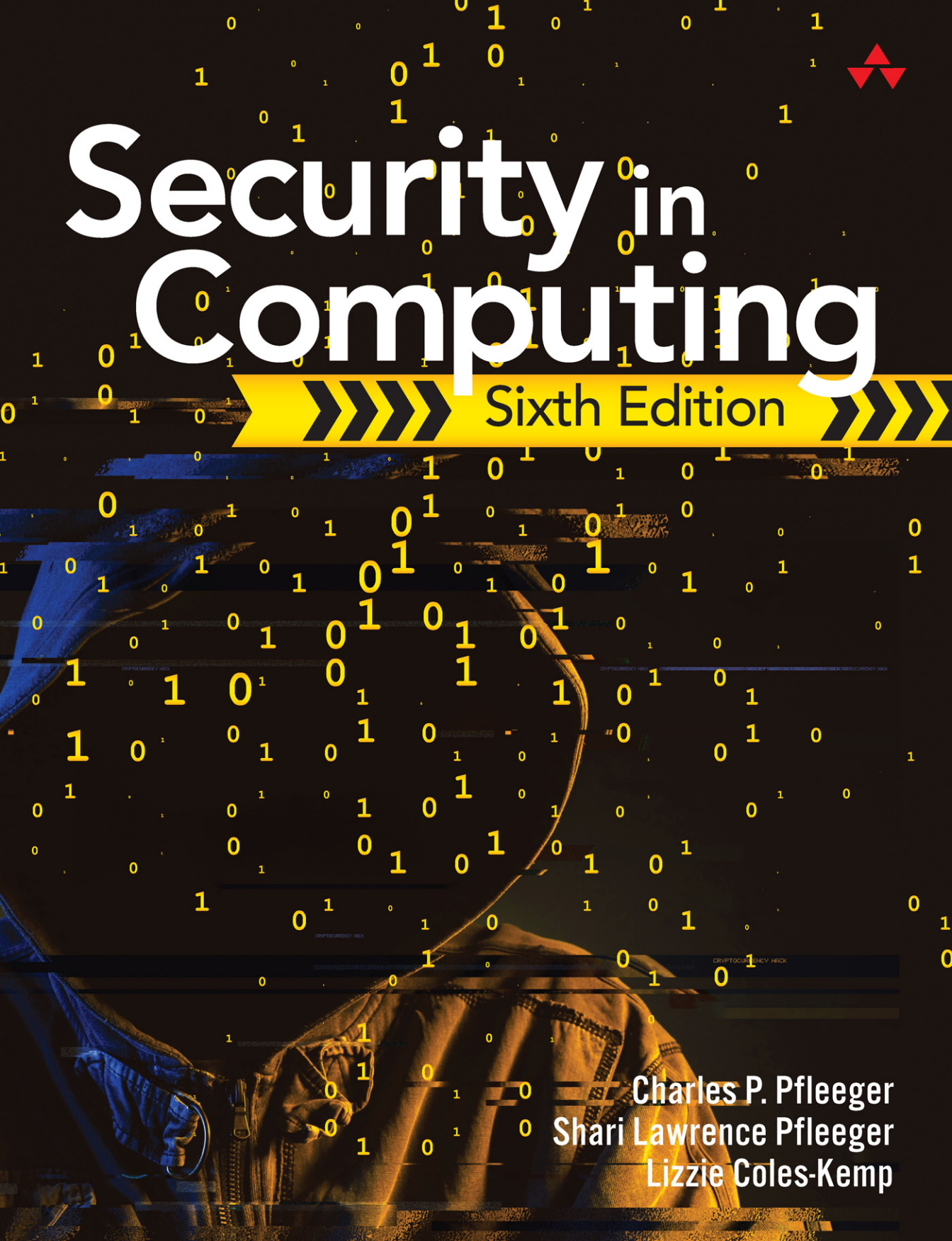
Security in Computing
Sixth Edition
Charles P. Pfleeger
Shari Lawrence Pfleeger
Lizzie Coles-Kemp

Security in Computing

SIXTH EDITION

Because so many people already have Adobe Reader installed, this example might not affect many machines. Suppose, however, the tool were a special application from a bank to enable its customers to manage their accounts online, a toolbar to assist in searching, or a viewer to display proprietary content. Many sites offer specialized programs to further their business goals, and, unlike the case with Adobe Reader, users will often not know whether the tool is legitimate, the site from which the tool comes is authentic, or the code is what the commercial site intends. Thus, website modification has advanced from being an attention-seeking annoyance to a serious potential threat.

Later in this chapter we consider mobile apps, programs that run on mobile devices such as phones or tablet computers. Commercial sites such as retailers, banks, and news media provide apps to display their content and help their customers. Users have few ways to determine whether an app is legitimate.

Web Bug

You probably know that a webpage is made up of many files: some text, graphics, executable code, and scripts. When the webpage is loaded, a browser downloads and processes files from a destination; during the processing, these files may invoke other files (perhaps from other sites) that in turn are downloaded and processed, until all invocations have been satisfied. When a remote file is fetched for inclusion, the request also sends the IP address of the requester, the type of browser, and the content of data files called **cookies** stored for the requested site. A cookie is a data file containing any data the page owner wants, in any format, including being encrypted. Using a cookie, the webpage server might display "Welcome back, Zoé," bring up content from the last visit, or open at a particular webpage. Cookies preserve continuity between website visits, even if the visits happen days or weeks apart.

Some advertisers want to count the number of visitors and number of times each visitor arrives at a site. They can do this by a combination of cookies and an invisible image; this image is a form of steganography, something hidden in plain sight, as described in Chapter 3.

A **web bug**, also called a **clear GIF, 1x1 GIF**, or **tracking bug**, is a tiny image, as small as 1 pixel by 1 pixel. (Depending on resolution, screens display at least 100 to 200 pixels per inch.) A 1x1 image is so small it will not normally be seen. Nevertheless, the image is loaded and processed just as a larger picture would be. Part of the processing is to notify the bug's owner, the advertiser, of its display on another user's screen.

A single company can do the same thing without needing a web bug. If you order flowers online, the florist can obtain your IP address and set a cookie containing your details so as to recognize you in the future as a repeat customer. A web bug allows this tracking across multiple merchants.

Your florist might subscribe to a web tracking service; for this example, we will call it ClicksRUs. The florist includes a web bug for ClicksRUs among the images on its webpage. When you load that page, your details are sent to ClicksRUs, which then installs a cookie. If you leave the florist's website and next go to a bakery's site that also subscribes to tracking with ClicksRUs, the new page will also have a ClicksRUs

web bug. This time, as shown in Figure 4-13, ClicksRUs retrieves its old cookie, finds that you were last at the florist's site, and records the coincidence of these two firms. After correlating these data points, ClicksRUs can inform the florist and the bakery that they have a common customer; then, the two business might, for example, develop a joint marketing approach. Or ClicksRUs can determine that you went from florist A to florist B to florist C and back to florist A, so it can report to B and C that they lost out to A, helping them all develop more successful marketing strategies. Or ClicksRUs can infer that you are looking for a gift and will offer a targeted ad on the next site you visit. ClicksRUs might receive advertising revenue from florist D and trinket merchant E, which would influence the ads it will display to you. Web bugs and tracking services are big business, but they are also threats to privacy, as we explain in Chapter 9.

Tiny action points called web bugs can report page traversal patterns to central collecting points, compromising privacy.

Web bugs can also be used in email with images. A spammer gets a list of email addresses but does not know whether the addresses are active, that is, if anyone reads mail at those addresses. With an embedded web bug, the spammer receives a report when someone opens the email message on a browser.

Is a web bug malicious? Probably not, although some people would claim that the unannounced tracking is a harmful invasion of privacy. But invisible images are also useful in more malicious activities, as described next.

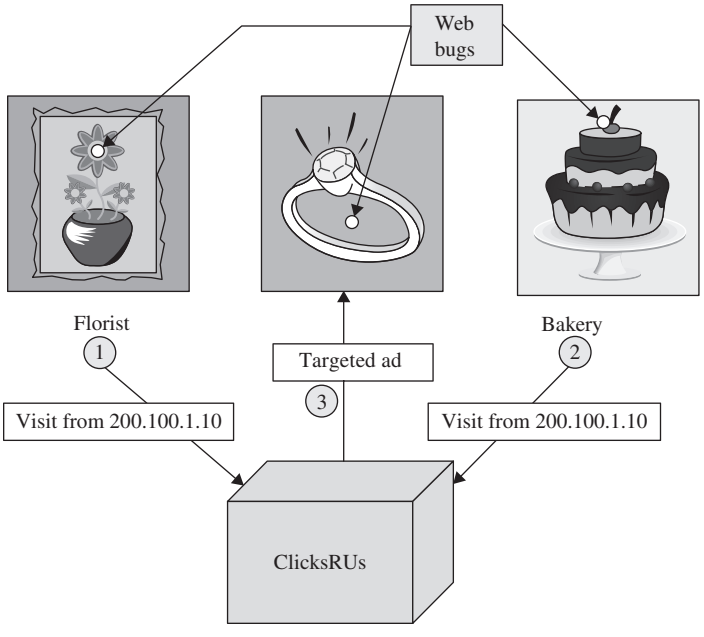

FIGURE 4-13 Web Bugs

Clickjacking

Suppose you are at a gasoline filling station with three buttons to press to select the grade of fuel you want. The station owner notices that most people buy the lowest-priced fuel but that the greatest profit comes from the highest-priced product. The owner decides to pull a trick by pasting stickers over the buttons for the lowest and highest prices reading, respectively, "high performance" (on the lowest-priced button) and "economy" (on the expensive, high-profit button). Thus, some people will push the economy (high-priced) button and unwittingly generate a higher profit for the owner. Unfair and deceptive, yes, but if the owner is unscrupulous, the technique would work; however, most businesses would not try that because it is unethical and might lose customers. Computer attackers do not care about ethics or loss of customers, so a version of this technique becomes a computer attack.

Consider a scenario in which an attacker wants to trick a victim into doing something. As you have seen in several examples in this book, planting a Trojan horse is not difficult. But good application programs and the operating system require a user to confirm actions that are potentially dangerous—the equivalent of a gas pump display that would ask "Are you *sure* you want to buy the most expensive fuel?" The trick is to get the user to agree without realizing it.

As shown in Figure 4-14, this computer attack uses a neutral image pasted over—that is, displayed on top of—another more sinister image. We are all familiar with the click box "Do you want to delete this file? [Yes] [No]." **Clickjacking** is a technique that essentially causes that prompt box to slide around so that [Yes] is always under the mouse. The attacker also makes this (sinister) box transparent, so the victim cannot see it and is thus unaware of clicking anything. Furthermore, a neutral second, visible image is pasted underneath, so the victim thinks the box being clicked is something like "For a Free Prize, Click [Here]." The victim clicks where [Here] is on the screen, but [Here] is not a button at all; it is just a picture directly under the invisible, sinister [Yes] box. The mouse click selects the [Yes] button.

Clickjacking: tricking a user into clicking a link by disguising what the link points to

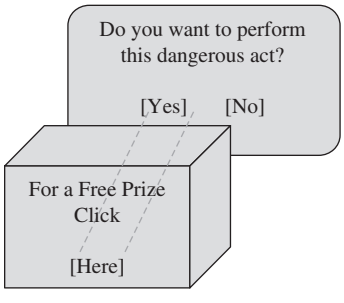

FIGURE 4-14 Clickjacking

It is easy to see how this attack would be used. The attacker chooses an action to which the user would ordinarily not agree, such as

- Do you really want to delete all your files?
- Do you really want to send your contacts list to a spam merchant?
- Do you really want to install this program?
- Do you really want to change your password to AWordYouDontKnow?
- Do you really want to allow the world to have write access to your profile?

For each such question, the clickjacking attacker only has to be able to guess where the confirmation box will land, make it transparent, and slip the "For a Free Prize, Click [Here]" box under the invisible [Yes] button of the sinister action's confirmation box.

These examples give you a sense of the potential harm of clickjacking. A surveillance attack might activate a computer camera and microphone, and the attack would cover the confirmation box; this attack was used against Adobe Flash, as shown in the video at youtube.com/watch?v=gxyLbpldmuU. Sidebar 4-8 describes how numerous Facebook users were duped by a clickjacking attack.

SIDEBAR 4-8 Facebook Clickjack Attack

In summer 2010, thousands of Facebook users were tricked into posting that they "liked" a particular site. According to BBC News (3 June 2010), victims were presented with sites that many of their friends had "liked," such as a video of the World Cup. When the users clicked to see the site, they were presented with another message asking them to click to confirm they were over age 18.

What the victims did not see was that the confirmation box was a sham underneath an invisible box asking them to confirm they "liked" the target website. When the victims clicked that they were over 18, they were really confirming their "like" of the video.

This attack seems to have had no malicious impact, other than driving up the "like" figures on certain websites. You can readily imagine serious harm from this kind of attack, however.

A clickjacking attack succeeds because the attacker can

- cause the browser to load a page with a confirmation button that commits the user to an action with one or a small number of mouse clicks (for example, "Do you want to install this program? [Yes] [Cancel]")
- change the confirmation box's image coloring to transparent
- move the image to any position on the screen
- superimpose a neutral image underneath the (transparent) sinister image with what looks like a button directly under the real (but invisible) button for the action the attacker wants (such as, "Yes" install the program)
- induce the victim to click what seems to be a button on the neutral image

The two technical tasks, changing the color to transparent and moving the page, are both possible because of a technique called **framing**, or using an **iframe**. An iframe is

a structure that can contain all or part of a page, can be placed and moved anywhere on another page, and can be layered on top of or underneath other frames. Although important for managing complex images and content, frames also facilitate clickjacking.

But, as we show in the next attack discussion, the attacker can obtain or change a user's data without needing to create complex web images.

Drive-By Download

Similar to the clickjacking attack, a **drive-by download** is an attack in which code is downloaded, installed, and executed on a computer without the user's permission and usually without the user's knowledge. In one example of a drive-by download, in April 2011 a webpage from the U.S. Postal Service was compromised with the Blackhole commercial malicious-exploit kit. Clicking a link on the postal service website redirected the user to a website in Russia, which presented what looked like a familiar "Error 404—Page Not Found" message, but instead the Russian site installed malicious code carefully matched to the user's browser and operating system type (*eWeek*, 10 April 2011).

Drive-by download: downloading and installing code other than what a user expects

Eric Howes [HOW04] describes an attack in which he visited a site that ostensibly helps people identify lyrics to songs. Suspecting a drive-by download, Howes conducted an experiment in which he used a sacrificial computer for which he had a catalog of installed software, so he could determine what else had been installed after visiting the website.

On his entry, the site displayed a pop-up screen asking for permission to install the program "software plugin" from "Software Plugin, Ltd." The pop-up was generated by a hidden frame loaded from the site's main page, seeking to run the script download-mp3.exe, a name that seems appropriate for a site handling music. When he agreed to the download, Howes found eight distinct programs (and their support code and data) downloaded to his machine.

Among the changes he detected were

- eight new programs from at least four different companies
- nine new directories full of files
- three new browser toolbars (including the interesting toolbar shown in Figure 4-15)
- numerous new desktop icons
- an addition to the bottom of the Save As dialog box, offering the opportunity to buy a computer accessory and take part in a survey to enter a sweepstakes
- numerous new Favorites entries
- a new start page on his browser

FIGURE 4-15 Drive-By Downloaded Toolbar

Removing this garbage from his computer was a challenge. For example, undoing the browser start page worked only while the browser was open; closing the browser and reopening it brought back the modified start page. The operating system listed only some of the junk programs in Add/Remove Programs, and removing programs that way was only partially successful. Howes also followed the paths to the companies serving the software and downloaded and ran uninstall utilities from those companies, again with only partial success. After those two attempts at removal, Howes's anti-malware utilities found and eradicated still more code. He finally had to remove a few stray files by hand.

Fortunately, it seems this attack planted no long-lasting, hidden registry changes that would have been even harder to eliminate. Howes was prepared for this download and had a spare machine he was willing to sacrifice for the experiment, as well as time and patience to undo all the havoc it created. Most users would not have been so prepared or so lucky.

This example hints at the range of damage a drive-by download can cause. Also, in this example, the user actually consented to a download (although Howes did not consent to all the things actually downloaded). In a more insidious form of drive-by download such as the postal service example, the download is just a script. It runs as a webpage, is displayed, and probes the computer for vulnerabilities that will permit later downloads without permission.

Protecting Against Malicious Webpages

The basic protection against malicious web content is access control, as presented in Chapter 2. In some way we want to prevent malicious content from becoming established or executed.

Access control accomplishes **separation**, keeping two classes of things apart. In this context, we want to keep malicious code off the user's system; alas, that is not easy.

Users download code to add new applications, update old ones, or improve execution. Additionally, often without the user's knowledge or consent, applications, including browsers, can download code either temporarily or permanently to assist in handling a data type (such as displaying a picture in a format new to the user). Although some operating systems require administrative privilege to install programs, that practice is not universal. And some naïve users run in administrative mode all the time. Even when the operating system does demand separate privilege to add new code, users tired of annoying pop-up boxes from the operating system routinely click [Allow] without thinking. As you can see, this explanation requires stronger action by both the user and the operating system, unlikely for both. The relevant measures here would include least privilege, user training, and visibility.

The other control is a responsibility of the webpage owner: Ensure that code on a webpage is good, clean, or suitable. Here again, the likelihood of that happening is small, for two reasons. First, code on webpages can be derived from many sources: libraries, reused modules, third parties, contractors, and original programming. Website owners focus on site development, not maintenance, so placing code on the website that seems to work may be enough to allow the development team to move on to the