Chapter 4
Web Client Security

Kannan Balasubramanian
Mepco Schlenk Engineering College, India

ABSTRACT

Although it is possible for a Web client to strongly authenticate a Web server and communicate privately with it (e.g., by using SSL and server-side certificates) not all security problems are solved. One reason is that access control management can only be really efficient for a small number of client-server relationships. Even in such a limited scenario, it requires some security expertise to recognize and manage good certificates. There are at least three good reasons for ensuring privacy and anonymity in the Web: to prevent easy creation of user profiles (e.g., shopping habits, spending patterns), to make anonymous payment systems possible, or to protect a company’s interests (e.g., information gathering in the Web can reveal its current interests or activities.

WEB CLIENT SECURITY ISSUES

The usefulness of the Web is in large part based on its flexibility, but that flexibility makes control difficult (Zwicky, et.al, 2000). Just as it’s easier to transfer and execute the right program from a web browser than from FTP, it’s easier to transfer and execute a malicious one. Web browsers depend on external programs, generically called viewers (even if they play sounds instead of showing pictures), to deal with data types that the browsers themselves don’t understand. (The browsers generally understand basic data types such as HTML, plain text, and JPEG and GIF graphics.) Netscape and Explorer now support a mechanism (designed to replace external viewers) that allows third parties to produce plug-ins that can be downloaded to become an integrated and seamless extension to the web browser. You should be very careful about which viewers and plug-ins you configure or download; you don’t want something that can do dangerous things because it’s going to be running on your computers, as if it were one of your users, taking commands from an external source. You also want to warn users not to download plug-ins, add viewers, or change viewer configurations, based on advice from strangers.

In addition, most browsers also understand one or more extension systems (Java, JavaScript, or ActiveX, for instance). These systems make the browsers more powerful and more flexible, but they also introduce new problems. Whereas HTML is primarily a text-formatting language, with a few extensions

DOI: 10.4018/978-1-5225-0273-9.ch004
for hypertext linking, the extension systems provide many more capabilities; they can do anything you can do with a traditional programming language. Their designers recognize that this creates security problems. Traditionally, when you get a new program you know that you are receiving a program, and you know where it came from and whether you trust it. If you buy a program at a computer store, you know that the company that produced it had to go to the trouble of printing up the packaging and convincing the computer store to buy it and put it up for sale. This is probably too much trouble for an attacker to go to, and it leaves a trail that’s hard to cover up. If you decide to download a program, you don’t have as much evidence about it, but you have some. If a program arrives on your machine invisibly when you decide to look at something else, you have almost no information about where it came from and what sort of trust you should give it.

The designers of JavaScript, VBScript, Java, and ActiveX took different approaches to this problem. JavaScript and VBScript are simply supposed to be unable to do anything dangerous; the languages do not have commands for writing files, for instance, or general-purpose extension mechanisms. Java uses what’s called a “sandbox” approach. Java does contain commands that could be dangerous, and general-purpose extension mechanisms, but the Java interpreter is supposed to prevent an untrusted program from doing anything unfortunate, or at least ask you before it does anything dangerous. For instance, a Java program running inside the sandbox cannot write or read files without notification. Unfortunately, there have been implementation problems with Java, and various ways have been found to do operations that are supposed to be impossible.

ActiveX, instead of trying to limit a program’s abilities, tries to make sure that you know where the program comes from and can simply avoid running programs you don’t trust. This is done via digital signatures; before an ActiveX program runs, a browser will display signature information that identifies the provider of the program, and you can decide whether or not you trust that provider. Unfortunately, it is difficult to make good decisions about whether or not to trust a program with nothing more than the name of the program’s source.

As time goes by, people are providing newer, more flexible models of security that allow you to indicate different levels of trust for different sources. New versions of Java are introducing digital signatures and allowing you to decide that programs with specific signatures can do specific unsafe operations. Similarly, new versions of ActiveX are allowing you to limit which ActiveX operations are available to programs. There is a long way to go before the two models come together, and there will be real problems even then.

Because an HTML document can easily link to documents on other servers, it’s easy for people to become confused about exactly who is responsible for a given document. “Frames” (where the external web page takes up only part of the display) are particularly bad in this respect. New users may not notice when they go from internal documents at your site to external ones. This has two unfortunate consequences. First, they may trust external documents inappropriately (because they think they’re internal documents). Second, they may blame the internal web maintainers for the sins of the world. People who understand the Web tend to find this hard to believe, but it’s a common misconception: it’s the dark side of having a very smooth transition between sites. Take care to educate users, and attempt to make clear what data is internal and what data is external.

Another common issue related to the use of the browser in the client side is the use of SSL encryption (http://www.w3.org/Security/Faq/wwwsf2.html). SSL uses public-key encryption to exchange a session key between the client and server; this session key is used to encrypt the http transaction (both request and response). Each transaction uses a different session key so that if someone manages to decrypt a