Web Security: A High Level View

Drew Dean
Dept. of Computer Science, Princeton University
35 Olden St., Princeton, NJ 08544
ddean@cs.princeton.edu

The explosive growth of the Internet has brought about a corresponding growth in security problems. While the IP protocol[1] has a provision for a security label in each datagram, this option has received little use. The meaning of the labels is only useful to the national security community, and no mechanism for tamper-proofing them is provided. In this abstract, we examine the HTTP protocol underlying the World Wide Web (WWW), discuss threats, and some of their solutions.

The HTTP/1.0 protocol[2] is remarkably simple. Each transaction opens a new TCP connection. There are 3 commands: `GET`, `POST`, and `PUT`. (`PUT` has received little use so far). `GET` retrieves a URL, and `POST` sends the information (presumably from a form), and reads the result. Meta-information is encoded in MIME headers, e.g., `Content-Type:`, `Content-Length:`, etc. This is about as simple as things can be. So what can go wrong? Plenty.

The first set of attacks to consider are spoofing attacks. There are many things that can be spoofed: DNS and IP addresses are the two most obvious. Executable content enables new content spoofing attacks. The good news is that we almost know how to prevent spoofing: the proper application of cryptographic techniques (cryptography, digital signatures, etc.) will solve the problem. Note the words “almost” and “proper” — these are critical. Peter Neumann’s work has shown that securely embedding cryptography into a system is a non-trivial job[3].

Let us now look at client-side security, i.e., what the user running Netscape Navigator, Microsoft Internet Explorer, or a similar user agent, needs to be concerned about. Ignoring executable content for the moment, the security requirement would seem to be quite simple: they merely display a document written in plain ASCII or HTML, a well documented standard[4]. Alas, it’s not quite so simple.

All of the common web browsers are written in unsafe programming languages, where the language provides no support for detecting buffer overflows, running off the end of an array, etc. It is a folk theorem that any crash due to an erroneous memory write (e.g., writing via a “smashed” pointer) is a potential security flaw. The question is: “How difficult is it for the attacker to control the value of the erroneous pointer?” It may be quite hard (and effectively unexploitable), or it may be quite simple (e.g., in case of a buffer overflow). Unfortunately, the current crop of web browsers seem to crash at least moderately often. It is unknown how many of these crashes are due to erroneous writes, but undoubtably some of them are. (Some others are due to erroneous reads, which may also be possible to exploit.) While these attacks may be difficult to exploit, they are clearly a threat for mission-critical systems (i.e., information warfare). These attacks are also a threat in the future: electronic commerce will mean that your computer can spend money. Thievery has motivated much more sophisticated attacks than these (as well as much more violence) throughout history, and there is no reason to believe that this will change.

If a web browser finds a `Content-Type:` it does not know how to handle, it will give the content to another (user-specified) "helper" program that knows how to interpret the data. Unfortunately, many of these programs were not written with security in mind. (That is, they never expected to be receiving untrusted input.) Bugs in these applications can lead to complete system penetration. Newer browsers also support "plug-ins" which are dynamically loaded into the browser, and execute in the same process as the browser. While one might hope that more attention is paid to security during their development, because the developers know that they will be receiving untrusted input, plug-ins pose an even bigger challenge to secure, because the underlying operating system has no way to differentiate them from the user's browser.

Executable content makes client-side security infinitely more complicated. We now have to consider the effects of an untrusted program running on the system. As pointed out by Steve Bellovin, this breaks an implicit assumption in most firewalls: that all the "bad guys" are on the outside. Confining executable content to a safe arena, while allowing for interesting applications is an open problem. Significant problems have been found in Java[5] and JavaScript. The all-or-nothing trust model of ActiveX may be good for large software vendors, but it offers the user little guidance whether they should trust software written by "Flight By Night Software and Dog Food, Inc." Alas, the joy of the WWW is that everyone can publish information on an equal footing: ActiveX does not effectively support this model.

Moving over to the Web server, we again find a seemingly simple program that turns out to be rather complicated in reality. Nominally, a HTTP server receives a request (in the form of a URL and other information encoded in MIME headers), and sends back some document (e.g., a HTML file, a text file, etc.). This is fairly simple, although the same sorts of bugs discussed above (buffer overflow, etc.) can happen here, with equally bad consequences. The complication comes in the Common Gateway Interface (CGI) support found in modern Web servers, along with server-side executable content.

The CGI architecture allows the author of a Web page to specify that a program will be run to generate the contents of a URL. While this is quite powerful (almost all forms feed their data back to a CGI program), it is also extremely dangerous. Examine the security situation here carefully: Assume that the Web server is run with the permissions of some system user (possibly, but not necessarily the superuser, but somewhat more privileged than an ordinary user). We now have the server executing a program written by a user who is semi-trusted at best, receiving input from a completely untrusted source. The external source could be attempting to penetrate the system, or the user could be attempting to gain additional privileges, or both. (Note that this situation is isomorphic to a mail transfer agent that will deliver mail to a program specified by the user. The history of sendmail is not particularly encouraging.) Present mass-market operating systems do not have useful confinement abilities for preventing either kind of abuse.

JavaSoft is developing a Web server, Jeeves, that will run client-specified Java bytecode "servlets." This seems inherently risky: your web server can't be quite as limited as a Java applet; if servlets can't query databases, they will not be very useful. At the same time, the set of queries a servlet can make needs to be restricted; the classic problems of database inference and aggregation rear their heads.

A rather bleak picture has been painted so far. What can be done about these problems? Executable content seems particularly difficult to support with present systems. A strong way to provide a strict subset of the user's privileges is required. This can either be implemented in the operating system, or in the executable content's runtime system (the approach Java attempts). The second approach has not been successful so far, but still has promise. A small, formally defined language, with proven type-soundness, should provide an interesting base to build a system on. (Java does not quite meet these criteria.)

As to the implementation of Web servers and user agents, this is a classic software engineering problem: we are really asking how to build reliable software. Safe languages, while not a panacea, relieve the programmer of many painful details that must be gotten correct. Formal methods have a place, too, in proving important theorems about high-level design issues, although formally proven implementations are not presently feasible. The payoff from formal methods is not just the proof; it is the bugs found that prevented the original proof attempt from going through. Unfortunately, the mass market is not yet interested in secure software; it is more interested in which browser has the most features and ships first. It seems that it will require a widely publicized, very expensive problem to change this attitude.

The original security situation in the WWW was conceptually simple: everything was static. While proper implementation was difficult, primarily due to the implementation technology used, the big picture was simple. Various executable content systems, along with CGI programs and servlets, have evolved both the client and server into complicated security situations, with insufficient support from the underlying programming languages and operating systems. Electronic commerce will provide the necessary motivation for sophisticated attacks. Cryptography does not address these issues; however, it appears to offer a solution to the spoofing problems, if properly deployed. Web security promises to be an active area for many years to come.

References

[1] Postel, Jon, "Internet Protocol," RFC 791, 1981.

[2] Berners-Lee, Tim, et al., "Hypertext Transfer Protocol," RFC 1945, 1996.

[3] Neumann, Peter, "Can Systems Be Trustworthy with Software-Implemented Cryptography?" Technical Report, Computer Science Laboratory, SRI International, Menlo Park, CA, October 1994.

[4] Berners-Lee, Tim, et al., "Hypertext Markup Language - 2.0," RFC 1866, November 1995.

[5] Dean, D., E. Felten, and D. Wallach, "Java Security: From HotJava to Netscape and Beyond," Proceedings of the 1996 IEEE Symposium on Security and Privacy, Oakland, CA, May 1996.