

Web Application Vulnerabilities

By Andrew Stern

Every year companies are moving more and more of their mission-critical applications and data into Web browsers. The reasons are obvious: browser-based applications are easier to roll out, support and maintain. As browsers and client scripting languages become more powerful and Web services become a standard, the trend is likely to continue. Indeed, even major industrial enterprise applications are now Web based.

Unfortunately, many of the features that make browsers so convenient and cost-effective also make them incredibly insecure. As a result, hackers are able to use Web based applications to penetrate corporate systems and access restricted customer records.

This article will explore some of the underlying reasons for these vulnerabilities in Web applications, illustrate some of the ways hackers commonly exploit them, and suggest a few ways in which companies can mitigate their risks.

The Root Causes of Application Vulnerability

Many of the most dangerous security holes in applications today are based not on worms or viruses, and not on known vulnerabilities in Application Servers, but on vulnerabilities in the *applications themselves*. These vulnerabilities—unique to each application—leave company Web infrastructures exposed to attacks such as cross-site scripting, SQL injections and cookie poisoning. An industry group called Open Web Application Security Project (OWASP) has produced an excellent list of some of these types of vulnerabilities, which they call their 'Top Ten Application Vulnerabilities.' (The list can be found at www.owasp.org) Other such lists exist—generally produced by security vendors to highlight their own particular strengths—but all highlight the same general types of vulnerabilities: browsers were simply never designed to validate the requests generated by users in an application.

Rather than list all the most frequent attacks, it might be useful to ask: *what is the common thread among all these attacks?* In other words, are these known attack methods just symptoms of some more fundamental limitations in Web applications?

The root cause of most of today's application vulnerabilities is their heritage: most Web applications were developed and tested using methodologies created for client-server applications. In some cases, they started out as client-server applications and were 'ported' to the Web, a process that often consisted of little more than reproducing the client GUI in a Web browser. In other cases, they were developed for the Web but their coding and testing practices did not incorporate the unique challenges of Web security.

Why is this a problem? It is a problem because Web applications have a distinct set of security challenges, challenges that are not obvious if you grew up in the client-server world.

Consider the paradigm of the old client-server development environment. The process of developing and testing client-server applications was modified, optimized and refined for decades, resulting in very robust applications. These applications consist of server code and client code written together and intended to work together. Deploying and maintaining client-server applications, however, remained very expensive: clients had to be tested on multiple operating systems, installed on every client machine, re-installed every time there was minor version change of the software, and so on. The cost of deploying new applications or maintaining existing ones for large numbers of users eventually became unmanageable.

To save money on deployment and maintenance, businesses have moved many if not most of their applications to the Web, often making minimal changes to the server while porting the client into a standard Web browser. Deployment is now free and clients are now platform-independent. However, the companies have exposed themselves to tremendous risk; the server code expects to have a trusted client but in fact has a Web browser client. In order to understand why that's so dangerous, let's look at the key differences between the two models.

Key Differences Between Client-Server and Web

Fundamental differences between Web applications and client-server applications open enterprises to significant risks when they move to the Web. For simplicity's sake, let's look at two major ways these applications differ.

Heavy Client vs. Browser

The heavy, purpose-built clients used in client-server applications are difficult to reverse-engineer, so it was difficult for hackers to modify inputs to the server. Think about it: when you have a piece of software running on your PC, you have no way to alter the messages it sends to its server—the software has fixed buttons and widgets, with no way to access the underlying code and manipulate the commands its generates. Browsers, however, are very easy to manipulate. The source of the client-side code is available to anyone accessing the Web page, and easy to change. (Users can simply go to "View Source" under the "View" menu of Internet Explorer or other browsers, change the code, and then reload the page.) In fact, some hacker Web sites now even offer tools such as 'Web proxies' which automate this type of manipulation.

To improve server performance, reduce traffic on the network, and enhance user experience, client-server applications do a lot of data validation on the client side. Web servers try to utilize the browser's capabilities (HTML and JavaScript) to do data validation on the client, but HTML can be changed and JavaScript can be disabled. In other words, any validation or

checks that developers can put into a browser, users can simply disable or change. This places the entire burden of input validation on the server side, and most programmers simply can't check every single input for potentially malicious values.

This aspect of Web applications exposes them to attacks such as Buffer Overflow, SQL Injections, and Cross-Site Scripting. For instance, users can take advantage of input fields in Web based forms to input malicious code. Oracle recently announced that Oracle Application 11i—their most Web-friendly enterprise software to date—had left several inputs un-validated, and open to tampering. An online auction site recently discovered a particularly nasty vulnerability in their e-commerce application: malicious users were listing auctions with dangerous JavaScript commands in the item descriptions. Since the auction software was not checking what people entered in to their descriptions, all these JavaScript commands were being served up to anyone who clicked on the auction. Through execution of their browsers, people were opening new windows, stealing cookies, and so on—a classic cross-site scripting attack.

State vs. No State

Let's look at the second major difference between the old client-server environment and the Web. In client-server environments, a continuous session is maintained between each user and the server. Once a user logs into the application, an unbroken connection feeds the user appropriate information. In Web environments there is no session; users request a page and then disappear from the server's view until another page is requested.

This opens the Web application to many types of attack. The most common is *forceful browsing* (also called *broken access control*). This attack consists of simply asking the Web server for pages or resources that you should not have access to. In some cases, it might be book-marking an internal page of an application, then jumping to that page later without first passing through an authentication page. A well-known financial services company recently had just this problem—users could literally bypass the login page if they knew the address of internal pages. In addition, sometimes users can just guess the URL of information they want to see. A major record label recently released a "sneak preview"—the first song of an upcoming album with a URL ending in /track1.mp3.

Obviously it wasn't long before the young audience realized that if they simply typed in "track2.mp3" or "track3.mp3" they could access the entire album. Somewhat more seriously, the Minnesota State Police Department recently discovered that anyone could append "/personsearch/personsearch.asp" to one of their public Web site addresses and access the department's police records. (They have since fixed the problem.)

Another inherent limitation of the Web's statelessness is that, in order to keep track of users, Web applications must leave something in the browser that identifies the user and the types of information requested. For instance, if the Web server leaves a hidden field on the page with a price or user group information, you can view the source of the page, change it, and then resubmit the page.

The most common way servers track users is session cookies. Again, a cookie resides on your PC, so you can access it and change it. Lee Gomes of the *Wall Street Journal* recently exposed this flaw at Gateway Computer. As Gomes reported, users could change their cookies and, when they went back to www.gateway.com, they were greeted with "Welcome back, x" and

were able to access x's credit card information, address the works. Change the cookie again and you would be 'recognized' as someone else!

What Can Be Done?


Today there are two ways to deal with these vulnerabilities. The first is to write explicit safeguards into the application code itself. There are no shortage of consulting firms providing training and frameworks for so-called "safe coding" initiatives and they are making applications much better than they have been historically. In addition, application development platforms are introducing more tools to help developers do things like validate inputs to prevent SQL injections and the like. Microsoft has introduced Web Form validation controls and Java now has tools such as Struts to provide similar capabilities. Of course, all of these tools are only as good as the developers implementing them, and are useless if they are not enforced. Furthermore, it's impossible to code against platform vulnerabilities. For instance, forceful browsing to a known or likely address can access a system or configuration file left exposed on a Web server.

Furthermore, most firms have dozens if not hundreds of legacy Web applications that are rife with vulnerabilities. An excellent place to begin making your application better is with scanning tools (SPI Dynamics' WebInspect product is my favorite). These can help to identify vulnerabilities in applications at both the platform and code level.

However, an ideal solution to the problem would offload application layer security to a centralized device where it could be easily managed and audited. This device would provide cost-effective, high-performance, pro-active application layer security from both generalized and targeted application attacks.

These products, commonly called Application Firewalls, are a reverse proxy that sits in front of Web servers and checks each request for malicious content or forceful browsing, passing only legitimate traffic to the Web servers behind them. They can block against vulnerabilities discovered by scanners like WebInspect (shared using AVDL, an open-source application vulnerability description language), or they start from scratch and create their own map of the application, blocking all non-recognized requests.

There is no 'magic bullet' to the problem of Web application vulnerabilities. A layered approach is required including better coding, tighter integration of development and security infrastructure, and application-aware network devices to do some of the heavy lifting.

Unfortunately, the problem of Web application vulnerabilities is not going away. Indeed, as enterprises become more tightly locked down at the network level and continue to put more information on the Web, application-level attacks will become increasingly common. Already an abundance of hacker Web sites provide tutorials and even shareware programs to facilitate this activity. Enterprises, in turn, must build out their own defenses. Until they have an application security strategy, they are no more secure than a castle with the front gate wide open. 

Andrew Stern is Director of Security Product Marketing for F5 Networks. He brings ten years of technology marketing and business development expertise to F5 Networks. He is a recognized expert in the web security industry and frequently lectures and authors articles on the subject.