

## Chapter 16

# Web Applications

---

### *In This Chapter*

- ▶ Attacking Web applications
  - ▶ Countering application hacking
- 

**W**eb applications, like e-mail, are common hacker targets because they're everywhere and often open for anyone to poke around in. Basic Web sites used for marketing, contact information, document downloads, and so on are a common target for hackers — especially the script-kiddie types — to deface. However, for criminal hackers, Web sites that store valuable information, like credit-card and Social Security numbers, are especially attractive. This is where the money is, so to speak.

Why are Web applications so vulnerable? The general consensus is they're vulnerable because of poor software development and testing practices. Sound familiar? It should, because this is the same problem that affects operating systems and practically all computer systems. This is the side effect of relying on software compilers to perform error checking, lack of user demand for higher-quality software, and emphasizing time-to-market instead of security and stability.

This chapter presents Web application hacks to check on your systems. You can test for literally thousands of vulnerabilities, but I focus on the ones I see most often. I also outline countermeasures to help minimize the chances that a hacker can carry out these attacks against your Web applications.

## *Web-Application Vulnerabilities*

Hacker attacks against insecure Web applications — via Hypertext Transfer Protocol (HTTP) — make up the majority of all Internet-related attacks. Most of these attacks can be carried out even if the HTTP traffic is encrypted (via HTTPS or HTTP over SSL) because the communications medium has nothing to do with these attacks. The security vulnerabilities actually lie within either the Web applications themselves or the Web server and browser software that the applications run on and communicate with.

Many attacks against Web applications are just minor nuisances or may not affect confidential information or system availability. However, some attacks can wreak havoc on your systems. Whether the Web attack is against a basic brochureware site or against the company's most critical customer server, these attacks can hurt your organization.

## *Choosing Your Tools*

Freeware and commercial tools can help ensure that your tests are comprehensive and minimize your testing time. All these tools basically work the same way, with such capabilities as scanning for script vulnerabilities, testing for invalid user input, and viewing critical files.

My favorite tools are Nikto ([www.cirt.net/code/nikto.shtml](http://www.cirt.net/code/nikto.shtml)), Nessus ([www.nessus.org](http://www.nessus.org)), and SPI Dynamics' WebInspect ([www.spidynamics.com](http://www.spidynamics.com)). These certainly are not the only tools available. It's still a young market for commercial tool vendors, so keep your eyes peeled for emerging products.

## *Insecure Login Mechanisms*

Many Web sites require users to login before they can do anything with the application. These login mechanisms often do not handle incorrect user IDs or passwords gracefully. They often divulge too much information that a hacker can use to gather valid user IDs and passwords.

### *Testing*

To test for insecure login mechanisms, browse to your application and login in the following ways:

- ✓ Using an invalid user ID with a valid password
- ✓ Using an valid user ID with an invalid password
- ✓ Using an invalid user ID and password

After you enter this information, the Web application probably responds with a message like *Your user ID is invalid* or *Your password is invalid*. The Web application may also return a generic error message, such as *Your user ID and password combination is invalid* and, at the same time, return different error codes in the URL for invalid user IDs and invalid passwords, as shown in Figures 16-1 and 16-2.

## Case study in hacking Web applications with Caleb Sima

In this case study, Caleb Sima, a well-known penetration-testing expert, shared an experience performing a Web-application security test. Here's his account of what happened.

### The Situation

Mr. Sima was hired to perform a Web-application penetration test to assess the security of a well-known financial Web site. Equipped with nothing more than the URL of the main financial site, Mr. Sima set out to find what other sites existed for the organization and began by using Google to search for possibilities. He initially ran an automated scan against the main servers to discover any low-hanging fruit. This scan provided information on the Web-server version and some other basic information, but nothing that proved useful without further research. And while Mr. Sima performed the scan, neither the IDS nor the firewall noticed any of his activity! Then he issued a request to the server on the initial Web page, which returned some interesting information. The Web application appeared to be accepting many parameters, but as he continued to browse the site, he noticed that the parameters in the URL stayed the same. He decided to delete all the parameters within the URL to see what information the server would return when queried. The server responded with an error message describing the type of application environment.

Next, Mr. Sima performed a Google search on the application that resulted in some detailed documentation. He found several articles and tech notes within this information that showed him how the application worked and what default files might exist. In fact, the server had several of these default files. He used this information to probe the application further. He quickly discovered internal IP addresses, as well as what services the application was offering. Now that he knew exactly what version the

admin was running, he wanted to see what else he could find.

Mr. Sima continued to manipulate the URL from the application by adding & characters within the statement to control the custom script. This allowed him to capture all source codes files! He noted some interesting filenames, including `VerifyLogin.htm`, `ApplicationDetail.htm`, `CreditReport.htm`, and `ChangePassword.htm`. Then he tried to connect to each file by issuing a specially formatted URL to the server. The server returned a *User not logged in* message for each request and stated that the connection must be made from the intranet.

### The Outcome

Mr. Sima knew where the files were located and was able to sniff the connection and determine that the `ApplicationDetail.htm` file set a cookie string. With little manipulation of the URL, he hit the jackpot! This file returned client information and credit cards when a new-customer application was being processed. `CreditReport.htm` allowed him to view customer credit-report status, fraud information, declined-application status, and a multitude of other sensitive information. The lesson to be learned: Hackers can utilize many types of information to break through Web applications. The individual exploits in this case study were minor, but when combined, they resulted in severe vulnerabilities.

Caleb Sima was a charter member of the X-Force team at Internet Security Systems and the first member of the Penetration Testing team. He went on to co-found SPI Dynamics ([www.spidynamics.com](http://www.spidynamics.com)) and become its CTO, as well as director of SPI Labs, the application-security research and development group within SPI Dynamics.



**Figure 16-1:**  
A login error in the URL for an invalid user ID.



**Figure 16-2:**  
A login error in the URL for an invalid password.

In either case, this is bad news, because the application is telling you not only which parameter is invalid, but also which one is *valid*. This means that the hackers now know either a good user name or password — their work has been cut in half! If they know the username (which usually is easier to guess), they can simply write a script to automate the password-cracking process, and vice versa. They can also use a remote Web login-cracking tool, such as Brutus ([www.hoobie.net/brutus](http://www.hoobie.net/brutus)), to attempt to break in, using a preconfigured file with user IDs and passwords, or even use it to perform brute-force attacks.

## Countermeasures

You can implement the following countermeasures to prevent hackers from attacking weak login systems in your Web applications:

- ✓ Any login errors that are returned to the end user should be as generic as possible, saying something like Your user ID and password combination is invalid.
- ✓ The application should never return error codes in the URL that differentiate between an invalid user ID and invalid password, as shown in Figures 16-1 and 16-2.



If a URL message must be returned, the application should keep it as generic as possible. Here's an example:

```
www.your_Web_app.com/login.cgi?success=false
```

This URL message may not be as convenient to the user, but it helps hide the mechanism and the behind-the-scenes actions from a hacker.

## Directory Traversal

A directory traversal is a really basic attack, but it can turn up interesting information about a Web site. This attack is basically browsing a site and looking for clues about the server's directory structure.

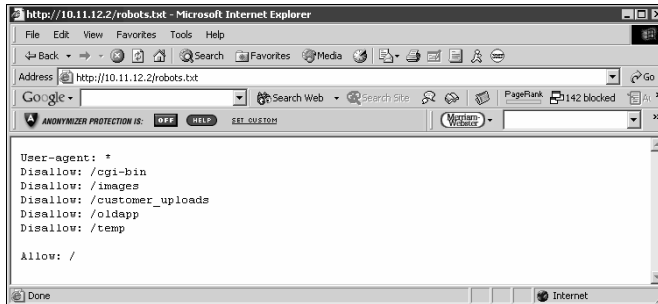
### Testing

Perform the following tests to determine information about your Web site's directory structure.

#### *robots.txt*

Start your testing with a search for the Web server's `robots.txt` file. This file tells search engines which directories not to index. Thinking like a hacker, you may deduce that the directories listed in this file may contain some information that needs to be protected. Figure 16-3 shows a `robots.txt` file that gives away information.

**Figure 16-3:**  
A Web server's robots.txt listing.



### Filenames

Confidential files on a Web server may have names like those of publicly accessible files. For example, if this year's product line is posted as `www.your_Web_app.com/productline2004.pdf`, confidential information about next year's products may be `www.your_Web_app.com/productline2005.pdf`.



A user may place confidential files on the server without realizing that they are accessible without a direct link from the Web site.

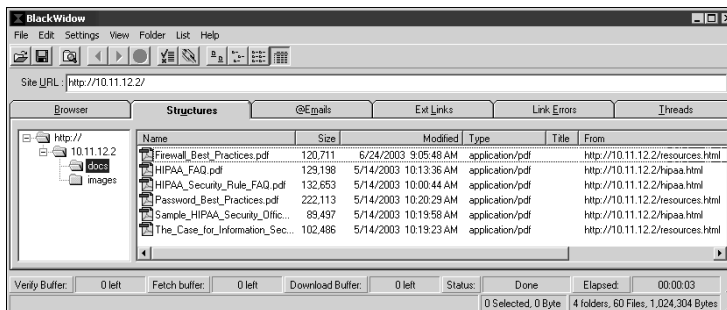
### Crawlers

A spider program like BlackWidow (`www.softbytelabs.com/BlackWidow`) can crawl your site to look for every publicly accessible file. Figure 16-4 shows the crawl output of a basic Web site.



Complicated sites often reveal more information that should not be there, including old data files and even application scripts and source code.

**Figure 16-4:**  
Using BlackWidow to crawl a Web site.



Look at the output of your crawling program to see what files are available. Regular HTML and PDF files are probably okay, because they're most likely needed for normal Web-application operation. But it wouldn't hurt to open each file to make sure it belongs.

## Countermeasures

You can employ two main countermeasures to having files compromised via malicious directory traversals:

- ✓ **Don't store old, sensitive, or otherwise nonpublic files on your Web server.** The only files that should be in your `/htdocs` or `DocumentRoot` folder are those that are needed for the site to function properly. These files should not contain confidential information that you don't want the world to see.
- ✓ **Ensure that your Web server is properly configured to allow public access only to those directories that are needed for the site to function.** Minimum necessary privileges are key here, so provide access only to the bare-minimum files and directories needed for the Web application to perform properly.

Check your Web server's documentation for instructions to control public access. Depending on your Web-server version, these access controls are set in

- The `httpd.conf` file and the `.htaccess` files for Apache  
Refer to [httpd.apache.org/docs/configuring.html](http://httpd.apache.org/docs/configuring.html) for more information.
- Internet Information Services Manager settings for Home Directory and Directory (IIS 5.1)
- Internet Information Services Manager settings for Home Directory and Virtual Directory (IIS 6.0)

The latest versions of these Web servers have good directory security by default, so if possible, make sure you're running the latest versions:

- ✓ Check for the latest version of Apache at [httpd.apache.org](http://httpd.apache.org).
- ✓ The most recent version of IIS (for Windows Server 2003) is 6.0.



## Input Filtering

Web applications are notorious for taking practically any type of input, assuming that it's valid, and processing it further. Not validating input is one of the greatest mistakes that Web-application developers can make. This can lead to system crashes, malicious database manipulation, and even database corruption.

## *Input attacks*

Several attacks can be run against a Web application that insert malformed data — often, too much at once — which can confuse, crash, or make the Web application divulge too much information to the attacker.

### *Buffer overflows*

One of the most serious input attacks is a buffer overflow that specifically targets input fields in Web applications.

For instance, a credit-reporting application may authenticate users before they're allowed to submit data or pull reports. The login form uses the following code to grab user IDs with a maximum input of 12 characters, as denoted by the `maxsize` variable:

```
<form name="Webauthenticate" action="www.your_Web_app.com/login.cgi"
      method="POST">
...
<input type="text" name="inputname" maxsize="12">
...
```

A typical login session would be presented a valid login name of 12 characters or less. However, hackers can manipulate the login form to change the `maxsize` parameter to something huge, such as 100 or even 1,000. Then they can enter bogus data in the login field. What happens next is anyone's call — they may lock up the application, overwrite other data in memory, or crash the server.

### *Automated input*

An automated-input attack is when a malicious hacker manipulates a URL and sends it back to the server, directing the Web application to add bogus data to the Web database, which can lead to various DoS conditions.

Suppose, for example, that you have a Web application that produces a form that users fill out to subscribe to a newsletter. The application automatically generates e-mail confirmations that new subscribers must respond to. When users receive their e-mail confirmations, they must click a link to confirm their subscription. Users can tinker with the hyperlink in the e-mail they received — possibly changing the username, e-mail address, or subscription status in the link — and send it back to the server hosting the application. If the Web server doesn't verify that the e-mail address or other account information being submitted has recently subscribed, the server will accept practically anyone's bogus information. The hacker can automate the attack and force the Web application to add thousands of invalid subscribers to its database. This can cause a DoS condition on the server or the server's network due to traffic overload, which can lead to other issues.



I don't necessarily recommend that you carry out this test in an uncontrolled fashion with an automated script you may write or download off the Internet. Instead, you may be better off carrying out this type of attack with an automated testing tool, such as WebInspect or, or one of its commercial equivalents, such as Sanctum's AppScan ([www.sanctuminc.com](http://www.sanctuminc.com)).

### **Code injection**

In a code-injection attack, hackers modify the URL in their Web browsers or even within the actual Web-page code before the information gets sent back to the server. For example, when you load your Web application from [www.your\\_Web\\_app.com](http://www.your_Web_app.com), it modifies the URL field in the Web browser to something similar to the following:

```
http://www.your_Web_app.com/script.php?info_variable=X
```

Hackers, seeing this variable, can start entering different data into the `info_variable` field, changing `X` to something like one of the following lines:

```
http:// www.your_Web_app.com/script.php?info_variable=Y
```

```
http:// www.your_Web_app.com/script.php?info_variable=123XYZ
```

The Web application may respond in a way that gives hackers more information — even if it just returns an error code — such as software version numbers and details on what the input should be. The invalid input may also cause the application or even the server itself to hang. Similar to the case study earlier in the chapter, hackers can use this information to determine more about the Web application and its inner workings, which can ultimately lead to a serious system compromise.



Code injection can also be carried out against back-end SQL databases — an attack known as *SQL injection*. Hackers insert rogue SQL statements to attempt to extract information from the SQL database that the Web application interacts with. Microsoft has a good Web site dedicated to Microsoft SQL Server security, including Slammer prevention and cleanup, at [www.microsoft.com/sql/techinfo/administration/2000/security/slammer.asp](http://www.microsoft.com/sql/techinfo/administration/2000/security/slammer.asp). Also check out the popular and effective Shadow Database Scanner at [www.safety-lab.com/en/products/6.htm](http://www.safety-lab.com/en/products/6.htm).

### **Hidden field manipulation**

Some Web applications embed hidden fields within Web pages to pass state information between the Web server and the browser. Hidden fields are represented in a Web form as `<input type="hidden">`. Due to poor coding practices, hidden fields often contain confidential information (such as product prices for an e-commerce site) that should be stored only in a back-end database. Users should not be able to see hidden fields — hence, the name — but the curious hacker can discover and exploit them with these steps:

**1. Save the page to the local computer.****2. View the HTML source code.**

To see the source code in Internet Explorer, choose View⇨Source.

**3. Change the information stored in these fields.**

For example, a hacker may change the price from \$100 to \$10.

**4. Re-post the page back to the server.**

This allows the hacker to obtain ill-gotten gains, such as a lower price on a Web purchase.

***Cross-site scripting***

Cross-site scripting (XSS) is a well-known Web application vulnerability that occurs when a Web page displays user input — via JavaScript — that isn't properly validated. A hacker can take advantage of the absence of input filtering and cause a Web site to execute malicious code on any user's computer that views the page.

For example, an XSS attack can display the user ID and password login page from another rogue Web site. If users unknowingly enter their user IDs and passwords in the login page, the user IDs and passwords are entered into the hacker's Web server log file. Other malicious code can be sent to a victim's computer and run with the same security privileges as the Web browser or e-mail application that's viewing it on the system; the malicious code could provide a hacker with full read/write access to the entire hard drive!



A simple test shows whether your Web application is vulnerable to XSS. Look for any parts of the application that accept user input (such as a login field or search field), and enter the following JavaScript statement:

```
<script>alert('You have been scripted')</script>
```

If a window pops up that says *You have been scripted*, as shown in Figure 16-5, the application is vulnerable.

---

**Figure 16-5:**  
A sample  
JavaScript  
pop-up  
window.

---



## Countermeasures

Web applications must filter incoming data. The applications must check and ensure that the data being entered fits within the parameters of what the application is expecting. If the data doesn't match, the application should generate an error and not permit the data to be entered. The first input validation of the form should be matched up with an input validation within the application to ensure that the input parameter meets the requirement.

Developers should know and implement these best practices:

- ✓ To reduce hidden-field vulnerabilities, Web applications should never present static values that the Web browser and the user don't need to see. Instead, this data should be implemented within the Web application on the server side and retrieved from a database only when needed.
- ✓ To minimize XSS vulnerabilities, the application should filter out `<script>` tags from the input fields.
- ✓ You can also disable JavaScript in the Web browser on the client side as an added security precaution.

Some secure software coding practices can eliminate all these issues from the get-go if they're made a critical part of the development process.

## Default Scripts

Poorly written Web programs, such as Common Gateway Interface (CGI) and Active Server Pages (ASP) scripts, can allow hackers to view and manipulate files on a Web server that they're not authorized to access, as well as upload tons of files that can eventually fill the Web server's hard drive. Attacks such as the Poison Null attack and Upload Bombing attack against vulnerable CGI scripts written in Perl permit unauthorized access.

## Attacks

Default script attacks are common because so many poorly written scripts are floating around the Internet. Hackers can also take advantage of various sample scripts that install on Web servers — especially older versions of Microsoft's IIS Web server.

Many Web developers and Webmasters use these scripts without understanding how they really work or testing them, which can introduce serious security vulnerabilities.





Some poorly written scripts contain confidential information, such as usernames and passwords! To test for this, you can peruse scripts manually or use a text search tool — such as the Search function built into the Windows Start menu or the find program in Linux or UNIX — to find any hard-coded usernames and passwords. Look for such words as *admin*, *root*, *user*, *ID*, *login*, *password*, *pass*, and *pwd*.

Confidential or critical information that's embedded in scripts like this is rarely necessary and is often the result of poor coding practices — convenience over security.

## Countermeasures

You can help prevent attacks against default Web scripts:

- ✓ Know how scripts work before deploying them within a Web application.
- ✓ Make sure that all default or sample scripts are removed from the Web server before using them.



Don't use scripts that have confidential information that's hard coded. They're a security incident in the making.

## URL Filter Bypassing

It's possible for internal employees to bypass Web-content filtering applications and logging mechanisms to browse to sites that they shouldn't go to — potentially covering up malicious behavior and Internet usage.

### Bypassing filters

Malicious employees bypass URL filtering mechanisms by using proxy servers, tunneling Web traffic over nonstandard ports, spoofing IP addresses, and so on. But an even-easier hack is to exploit the general mechanism built into URL filtering systems that filter Web traffic based on specific URLs and *keywords* (words that match a list or meet a certain criteria). Users take advantage of this practice by converting the URL to an IP address and then to its binary equivalent. The following steps can bypass URL filtering in such browsers as Netscape and Mozilla:

1. Obtain the IP address for the Web site.

For example, a gambling Web site ([www.go-gambling.com](http://www.go-gambling.com)) blocked in Web-content filtering software has this IP address:

```
10.22.33.44
```

This is an invalid public address, but it's okay for this example; you may want to filter out Web addresses on your internal network as well.

**2. Convert each individual number in the IP address to an eight-digit binary number.**

Numbers that may have fewer than eight digits in their binary form must be padded with leading zeroes to fill in the missing digits. For example, the binary number 1 is padded to 00000001 by adding seven zeroes.

The four individual numbers in the IP address in Step 1 have these equivalent eight-digit binary numbers:

```
10 = 00001010
```

```
22 = 00010110
```

```
33 = 00100001
```

```
44 = 00101100
```

The Windows Calculator can automatically convert numbers from decimal to binary notation:

- i. Choose View → Scientific.
- ii. Click the Dec option button.
- iii. Enter the number in decimal value.
- iv. Click the Bin option button to show the number in binary format.

**3. Assemble the four 8-digit binary numbers into one 32-digit binary number.**

For example, the complete 32-digit binary equivalent for 10.22.33.44 is

```
00001010000101100010000100101100
```

Don't add the binary numbers. Just organize them in the same order as the original IP address without the separating periods.

**4. Convert the 32-digit binary number to a decimal number.**

For example, the 32-digit binary number 00001010000101100010000100101100 equals the decimal number 169222444.

The decimal number doesn't need to be padded to a specific length.



5. Plug the decimal number into the Web browser's address field, like this:

```
http://169222444
```

The Web page loads easy as pie!



The preceding steps won't bypass URLs in Internet Explorer.

## Countermeasures

If the bypassing of certain Web-content filters is an issue for your network, ask your content-filtering vendor if it has a solution.

## Automated Scans

Automated application-security-assessment tools can find vulnerabilities within a Web application that are next to impossible to find otherwise.



You can't solely rely on automated tools to test your Web applications. But I can't imagine comprehensive security testing without them.

## Nikto

Figure 16-6 shows the partial results of a Nikto scan against a default IIS 5.0 installation. This scan found that the remote scripts directory is browsable and that the server is vulnerable to XSS. It also identified default scripts. Nikto found 16 potential vulnerabilities out of the 2,000 items checked.

## WebInspect

Figure 16-7 shows the output of a WebInspect scan against the default IIS 5.0 installation. This scan found XSS vulnerabilities, the IIS-specific Microsoft Data Access Components, and the `null.printer` vulnerabilities. WebInspect found a total of 208 potential vulnerabilities out of the 3,000 items checked.



## General Best Practices for Minimizing Web-Application Security Risks

Keeping your Web applications secure requires ongoing vigilance from an ethical hacking perspective and from your Web-application developers and vendors. Keep up with the latest hacks and testing tools and techniques.

### Obscurity

The following forms of security by *obscurity* can help prevent automated attacks from worms or scripts that are hard-coded to attack specific script types or default HTTP ports.



- ✓ To protect Web applications and related databases, use different machines to run each Web server, application, and database server.

The operating systems on these individual machines should be tested for security vulnerabilities and hardened based on best practices and the countermeasures. in Chapter 11 (Windows), Chapter 12 (Linux), and Chapter 13 (NetWare).

- ✓ Use built-in Web-server security features to handle access controls and process isolation, such as the application-isolation feature in IIS 6.0.

This helps ensure that if one Web application is attacked, it won't necessarily risk any other applications running on the same server.

- ✓ If you're concerned about platform-specific attacks being carried out against your Web application, you can trick the attacker into thinking the Web server or operating system is something completely different. Here are a few examples:

- If you're running a Microsoft IIS server and applications, you may be able to rename all your ASP scripts to have a `.cgi` extension.
- If you're running a Linux Web server, use a program such as IP Personality ([ippersonality.sourceforge.net](http://ippersonality.sourceforge.net)) to change the OS fingerprint so the system looks like it's running something else.

- ✓ Change your Web application to run on a nonstandard port. Change from the default HTTP port 80 or HTTPS port 443 to a high port number, such as 8877.



Don't rely on obscurity alone; it isn't foolproof. A dedicated hacker may be able to determine that the system isn't what it claims to be.

## Firewalls

Consider using these Web-application firewalls to protect your systems and information:

- ✓ A network-based firewall that can detect and block attacks against Web applications.
  - Commercial firewalls are available from such companies as NetScreen ([www.netscreen.com](http://www.netscreen.com)), TippingPoint Technologies ([www.tippingpoint.com](http://www.tippingpoint.com)), and Check Point ([www.checkpoint.com](http://www.checkpoint.com)).
  - An open-source firewall project called CodeSeeker is maintained by OWASP ([www.owasp.org/development/codeseeker](http://www.owasp.org/development/codeseeker)).
- ✓ Host-based Web-application intrusion-prevention applications such as
  - BlackICE — my all-time-favorite software application
  - Ubizen DMZ/Shield Enterprise ([www.ubizen.com](http://www.ubizen.com))
  - Eeye SecureIIS ([www.eeye.com](http://www.eeye.com))
  - McAfee Enterecept ([www.nai.com](http://www.nai.com))

These programs can detect Web-application attacks in real time and cut them off before they have a chance to do any harm.
- ✓ Find security holes in Web applications before they're deployed. Use a third-party code-examiner expert or an automated tool, such as Flawfinder ([www.dwheeler.com/flawfinder](http://www.dwheeler.com/flawfinder)), ITS4 ([www.digital.com/its4](http://www.digital.com/its4)), and RATS ([www.securesoftware.com/auditing\\_tools\\_download.htm](http://www.securesoftware.com/auditing_tools_download.htm))

Software development is where security holes begin and should end — but rarely do. If you can influence your Web developers, you can really make a difference in the security of your Web applications by encouraging secure development practices from the start. See Appendix A for resources.

