Web application security: Testing for vulnerabilities

Using open source tools to test your site

Skill Level: Intermediate

Jeff Orloff (jeff@sequoiamediaservices.com)
Consultant and Technology Coordinator
Sequoia Media, Inc.

20 Oct 2009

The increasing reliance on data-driven Web sites has caused an incline in the number of attacks launched against them. As a developer, understanding how a site can be attacked is paramount to making it secure. Discover some of the more common attacks, and learn about the tools you can use to spot them.

As the Web grows increasingly social in nature, inversely, it becomes less secure. In fact, the Web Application Security Consortium (WASC) estimated in early 2009 that 87% of all Web sites were vulnerable to attack (see Resources for links to more information). Although some companies can afford to hire outside security analysts to test for exploits, not everyone has the resources to spend US$20,000 to US$40,000 for an outside security audit. Instead, organizations become reliant on their own developers to understand these threats and make sure their code is devoid of any such vulnerability.

Frequently used acronyms

- **HTML:** Hypertext Markup Language
- **SQL:** Structured Query Language
- **URL:** Uniform Resource Locator

To write secure code, you must first understand the threats to which your work is exposed. This article looks at some of the more popular vulnerabilities, such as cross-site scripting and SQL injections, and introduces tools you can use to help safeguard not only your sites, but the data and networks that power them. This

article is not meant as a substitute for a security analyst or to teach hardcore security skills. Rather, it focuses on teaching you how to find potential exploits in your code and fix the vulnerabilities.

Common vulnerabilities

To begin, you need to understand some of the vulnerabilities you'll be looking for. The first is by far the most popular: cross-site scripting (XSS). XSS results from malicious scripts being injected into a Web site. For instance, Mallory writes a script that sends users to a trusted Web site created by Alice. Mallory inserts this script into a popular forum. When Bob sees this link on the forum, he clicks it and creates an account on Alice's Web site. The script, which has exploited an XSS flaw in Alice's Web site, then sends Bob's cookie to Mallory. Mallory can now impersonate Bob and steal information from him.

SQL injection is the second most popular vulnerability, primarily because of the growing dependence Web sites have on databases. SQL injection is actually quite simple: By finding a Web site that connects to a database, malicious hackers execute an SQL query in a place that the developer never intended for the purpose of bypassing authentication or manipulating data. This type of attack was what Albert Gonzalez used to steal 130 million credit card numbers. In an SQL injection attack, Mallory finds a Web site that Alice has created to sell electronics. Instead of entering the usual user name-password combination, Mallory enters `') OR 1=1-` for the user name. Because she included the hyphen (`-`), nothing else is required; because `1=1` is always true, she will be able to log in successfully. Now, she can manipulate the database to steal Bob's customer information. Although this example demonstrates an SQL injection in its simplest form, you can see just how easy it is for an attacker to use.

Without a specialized team of security analysts, it may seem impossible for the average Web developer to fight against these vulnerabilities. Fortunately, that is not the case. Many tools are available to help you find possible vulnerabilities on your sites so that you can take the necessary steps to protect against them. Tools like WebScarab and Paros capture conversations between the browser and server as well as crawl Web sites to help you identify potential risks. With this information in hand, you can then check these vulnerabilities and defend against them.

WebScarab

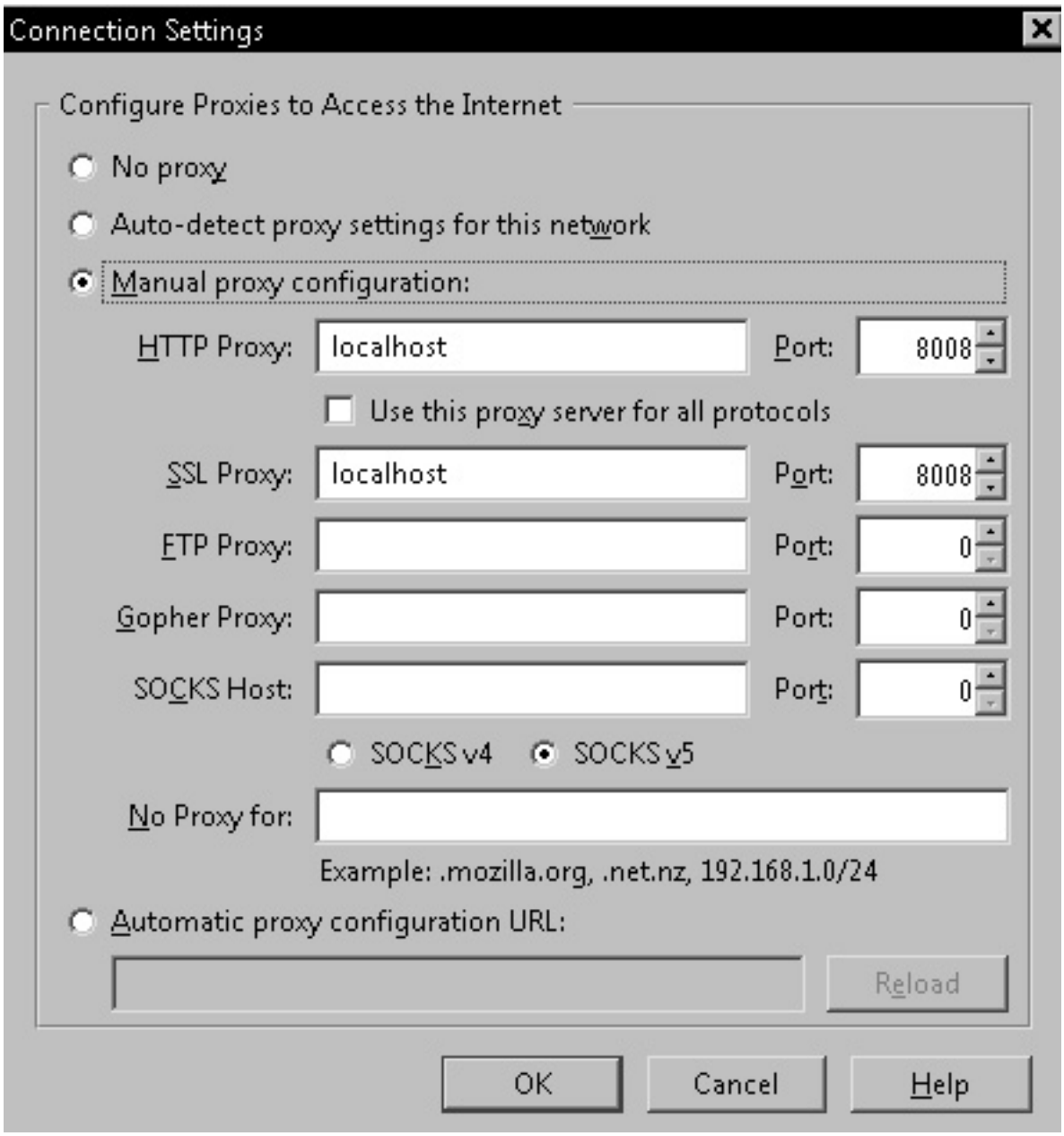
Developed by the Open Web Application Security Project (OWASP), WebScarab is first and foremost a proxy used to analyze browser requests and server replies. In addition to serving as a tool for packet analysis, you can use it to "fuzz" sites, looking for some of the same exploits mentioned above. To use WebScarab, you first configure proxy settings in your Web browser. For Mozilla Firefox, perform these

steps (see Firefox keyboard shortcuts for the alternative accessible steps):

1. Click **Tools > Options > Advanced > Network**.
2. Click **Settings**.
   The **Connection Settings** window opens.
3. Select the **Manual proxy configuration** option.
4. For both **HTTP Proxy** and **SSL Proxy**, type `localhost`, and set the port to 8008.
5. Make sure the **No proxy for** box is empty, then click **OK**.

Figure 1 shows Firefox's connection settings.

Figure 1. Firefox connection settings



For Windows® Internet Explorer®, perform the following steps (see Internet Explorer keyboard shortcuts in the IE Help menu for the alternative accessible steps):

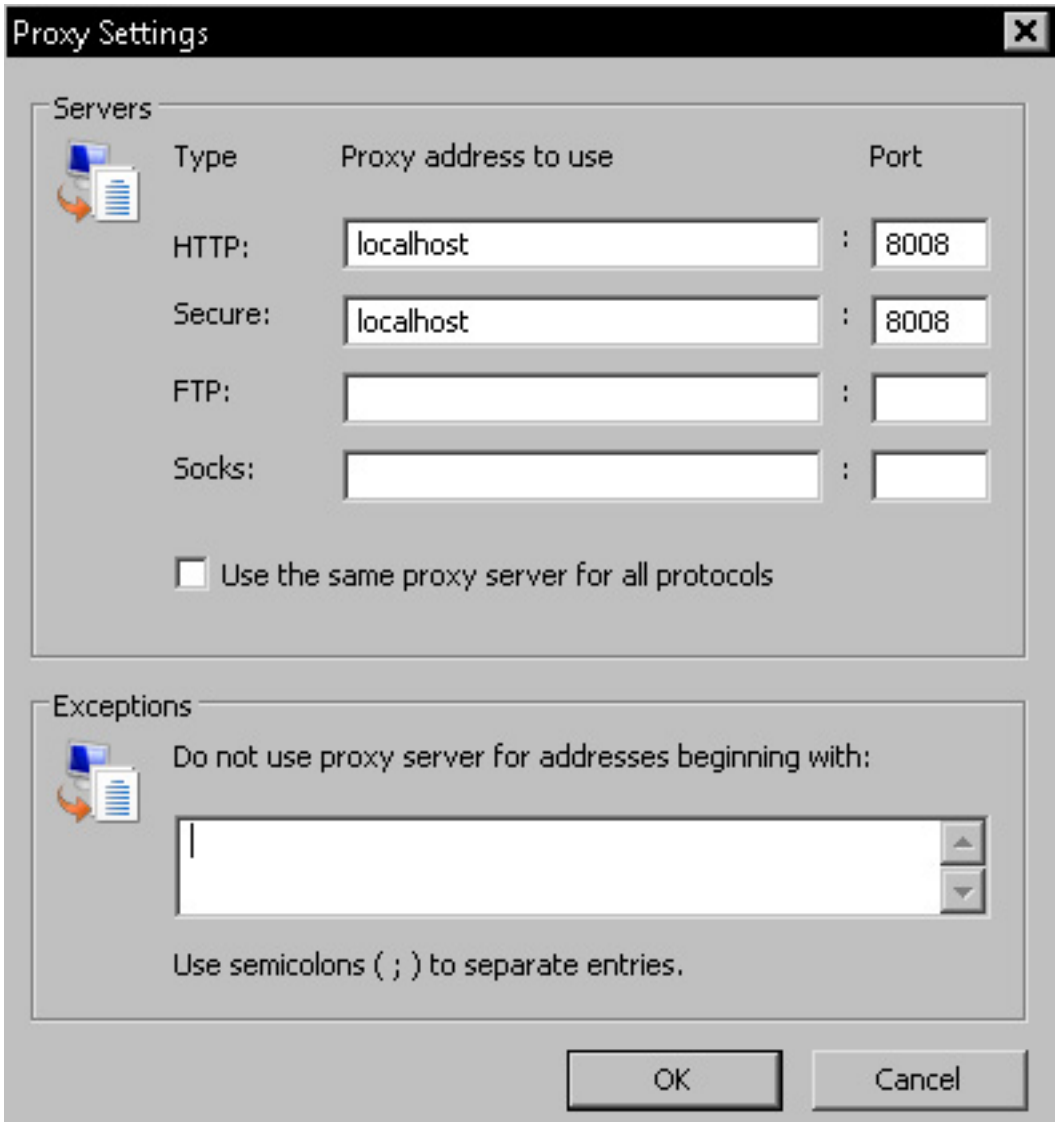
1. Click **Tools > Internet Options > Connections**.
2. Click **LAN Settings** to open the **Proxy Settings** window.
3. Under **Proxy Servers**, select the **Use a proxy server for your LAN** check box, then click **Advanced**.

4. For **HTTP** and **Secure**, type `localhost` for the proxy address to use, then type `8008` for the port.

5. Make sure the **Exceptions** box is empty, then click **OK**.

Figure 2 shows the proxy settings for Internet Explorer.

Figure 2. Setting the proxy



Now, you are ready to scan a Web site.

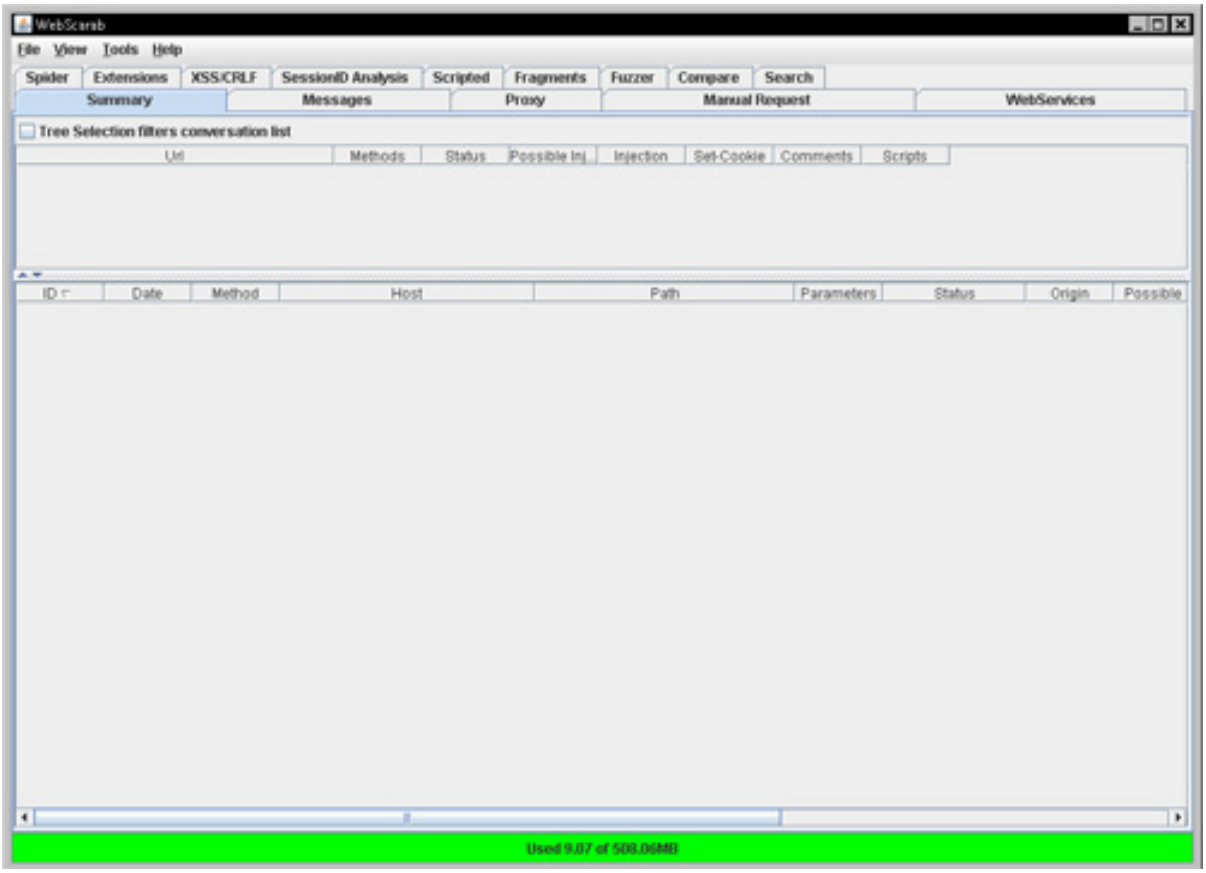
Scan your site with WebScarab

The following examples use the Hacme Casino site, which is built by Foundstone with certain vulnerabilities so that it can be used for training purposes (see

Resources). The site comes packaged with Apache Tomcat, as well, so you can run it locally if you wish.

Next, open WebScarab. Once you launch it, you will see only two tabs: **Summary** and **Intercept**. Because you want to use WebScarab for fuzzing, change the view by clicking **Tools > Use full-featured interface > OK**. You will then be asked to restart WebScarab. When you have done so, you can begin with the interface shown in Figure 3. (WebScarab provides a list of keyboard shortcuts—unfortunately only two.)

Figure 3. The WebScarab interface



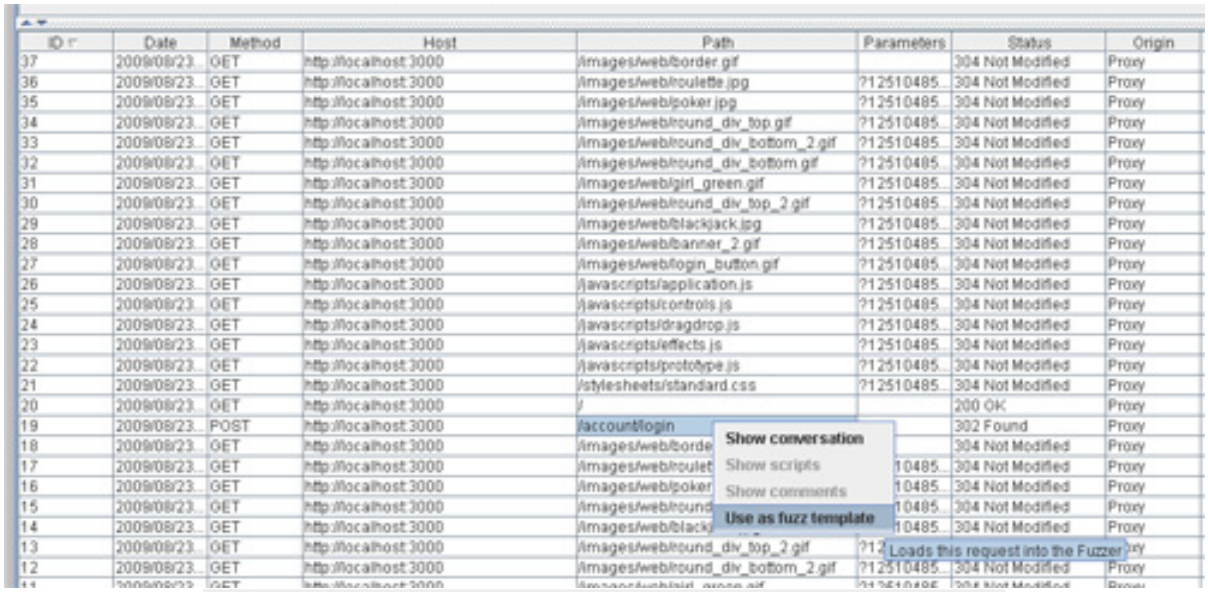
To start your new session, click **File > New**. You are presented with a box that asks where you would like to save your session. Select or create the storage directory, then click **OK**. When you use WebScarab with Hacme Casino, start the server before opening your browser. Then, in the browser's address bar, type `http://localhost:3000`.

Once you open the browser, you should start seeing some activity in WebScarab, because WebScarab is capturing all the requests and replies between the browser and the server. Because you want to use the fuzzing feature to test for vulnerabilities, first look at where the different vulnerabilities can exist. Start by looking at the login as a potential exploit. To test the login, you should record the interaction that takes place between the browser and the server. There is no need

for a successful login here, so you can enter whatever you like for the user name and password: For this example, enter `casino` for both the user name and password, then click **Login**.

Of course, the login will be unsuccessful, but that is not what you're interested in. Instead, go back to WebScarab. In the **Path** column, right-click **account\login**, then click **Use as fuzz template**, as shown in Figure 4.

Figure 4. The fuzzing template



Modifying the all_attack.txt file
Add the lines `'or 'x'='x` and `') or 1=1--` to the file, because they are basic SQL injection strings and will work in this example.

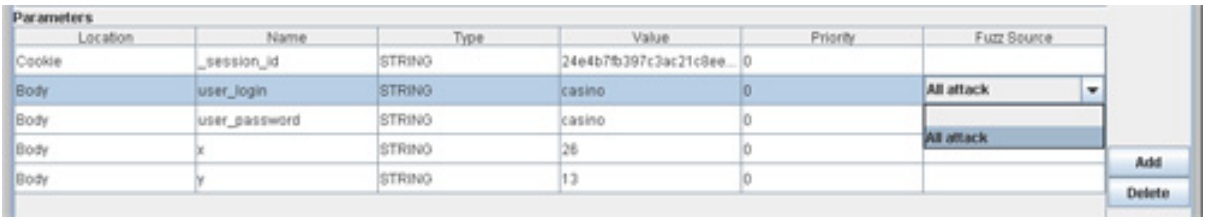
Perform the following steps:

1. Click the **Fuzzer** tab at the top of the WebScarab interface.
2. Click **Source**.
3. Browse to the location of your dictionary. (I used all_attack.txt; see Resources for a link.)
4. Provide a description of the source, then click **Add**.
5. Click **Close** to return to the **Fuzzer** window.

Now, go to the `user_login` parameter and click in the area under the **Fuzz Source** column. Because you have only loaded one source—All attack—select it from the

drop-down menu. Do the same for the `user_password` parameter, as shown in Figure 5.

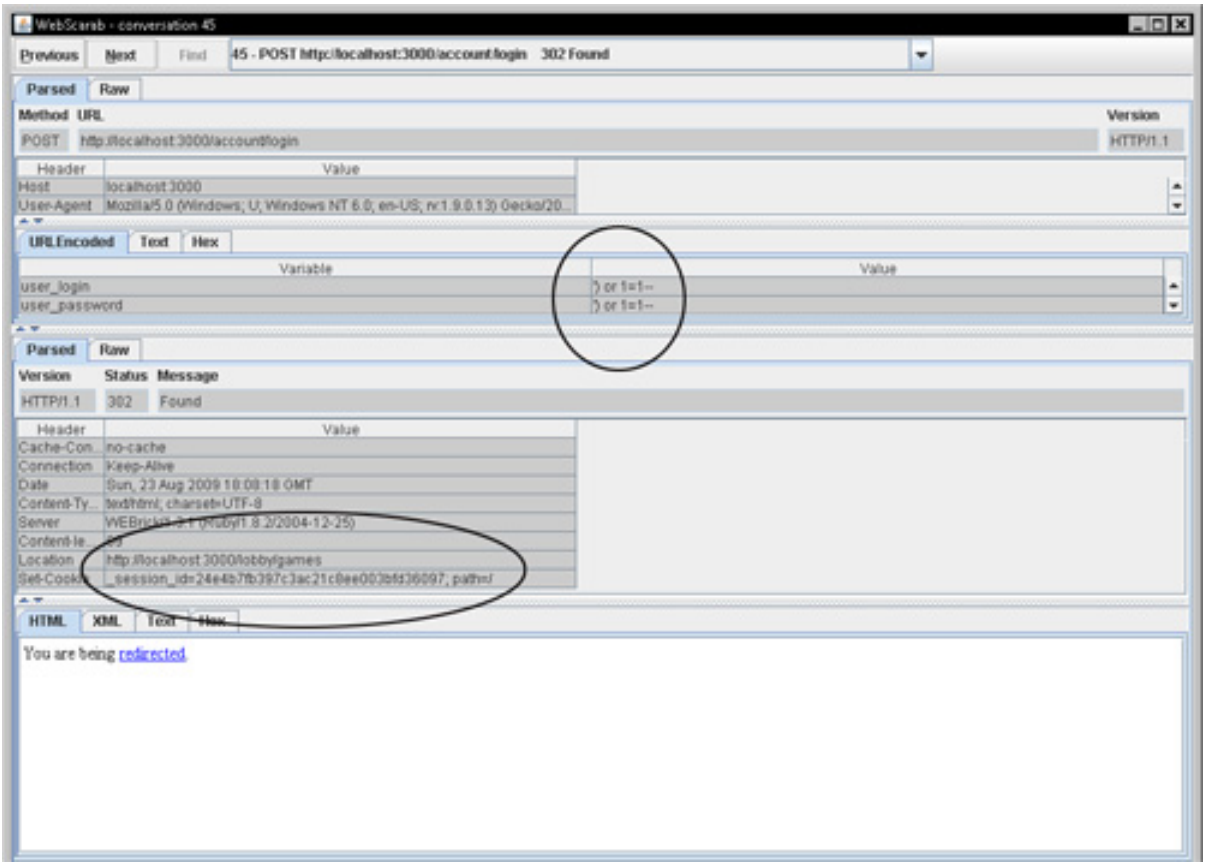
Figure 5. Selecting the attack file



Now that you have defined the source, click **Start** to begin the process. The fuzzer inserts the parameters from the file into both the user name and password inputs. You should see some activity on the screen as the tool completes each attempt. Typically, you would need to examine each ID in the results to analyze what takes place with each attempt. However, because you know that the added strings will produce a result, you can see what can happen when the fuzzer is successful.

Double-click the first string in the example, shown in Figure 6.

Figure 6. Finding a vulnerability



Note that in the top circle, the value for `user_login` and `user_password` is `') OR 1=1--`. Next, note that the location has changed from

`http://localhost:3000`, which is the home page, to `http://localhost:3000/lobby/games`, which is what a user who successfully logged in would see. What does this mean? It means that your site is vulnerable. Because `') OR 1=1--` is an SQL string, the site is vulnerable to an SQL injection. Scroll through the results using the **Next** button, and look for that value. Notice that the location here remains `http://localhost:3000`. You know from this result that inserting this type of string would not result in a successful login attempt, so the site is not vulnerable to this type of attack.

Paros Proxy

Another useful tool for security testing is Paros Proxy. Like WebScarab, Paros captures conversations between the browser and the server for analysis. You can also use it to test for vulnerabilities on a site. To run Paros, you must change the port number used in the proxy settings of your browser. For WebScarab, you used 8008; change this port to 8080 before running Paros, or the tool won't work. For this exercise, the example uses WebGoat, another tool from OWASP (see Resources). Like Hacme Casino, WebGoat runs using the Tomcat server as a local host. For the purposes of this article, WebGoat shows more vulnerabilities in the Paros scan than would Hacme Casino.

> **Using multiple scanners**
> Anyone testing his or her own sites should learn from this that different scanners may show different results for the same site. That is why professional pen-testers use multiple tools when working.

After changing the port, open Paros. Then, perform these steps:

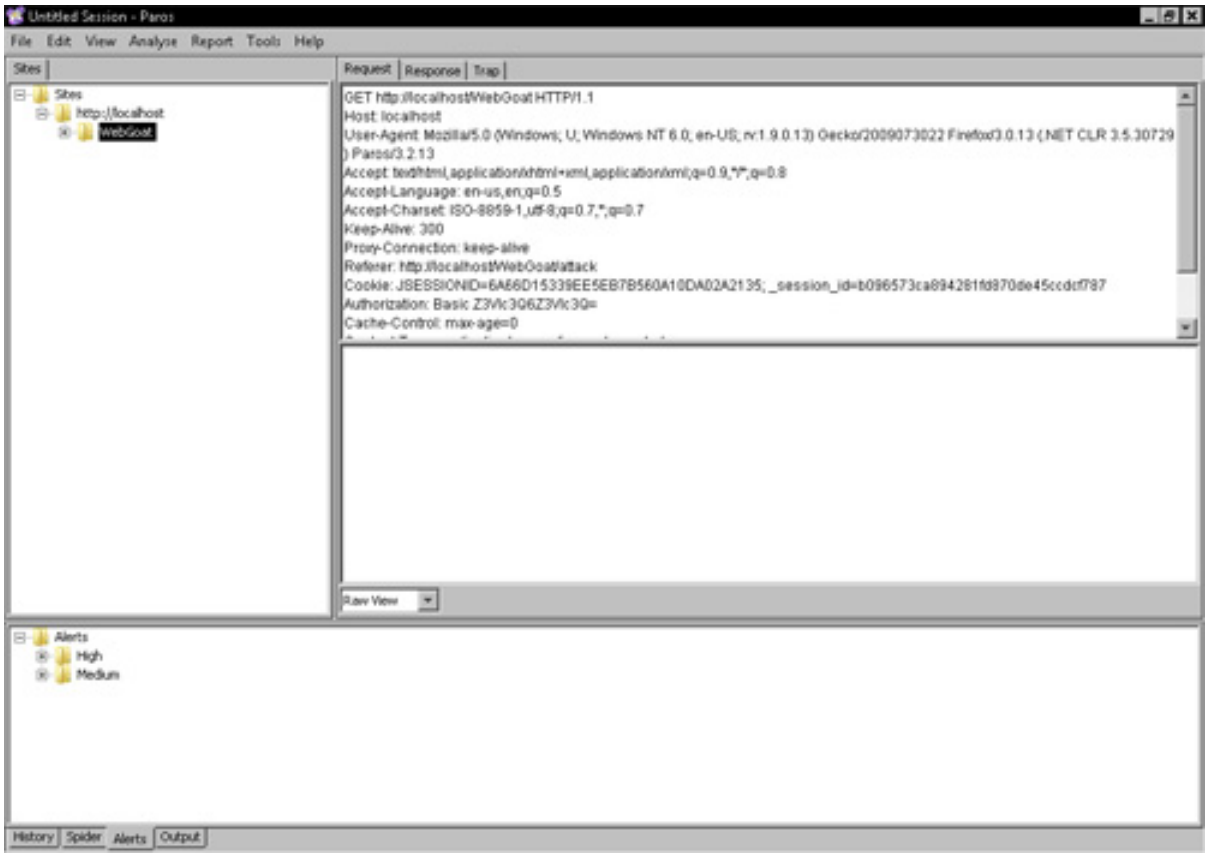
1. In the WebGoat folder, double-click **WebGoat.ba** to start Tomcat.
2. Open your browser, then type `http://localhost/WebGoat/attack`.
3. Type `guest` for the user name and password.
4. Click **Start WebGoat**.

Now, go back to Paros and start your scan of the WebGoat site. Of course, you can use your own site instead by entering its URL in the address bar.

In Paros, expand the Sites folder to see http://localhost in the tree. Expand this folder to bring up WebGoat. Highlight WebGoat, and then click **Analyse Scan**. This immediately begins the scan of the site. Large sites may take a while, but this one should only take a few seconds. Once it starts scanning, the bottom pane changes to the **Alerts** tab automatically (see Figure 7). When the scan is complete, click **OK**

in the pop-up window that tells you where to find the report.

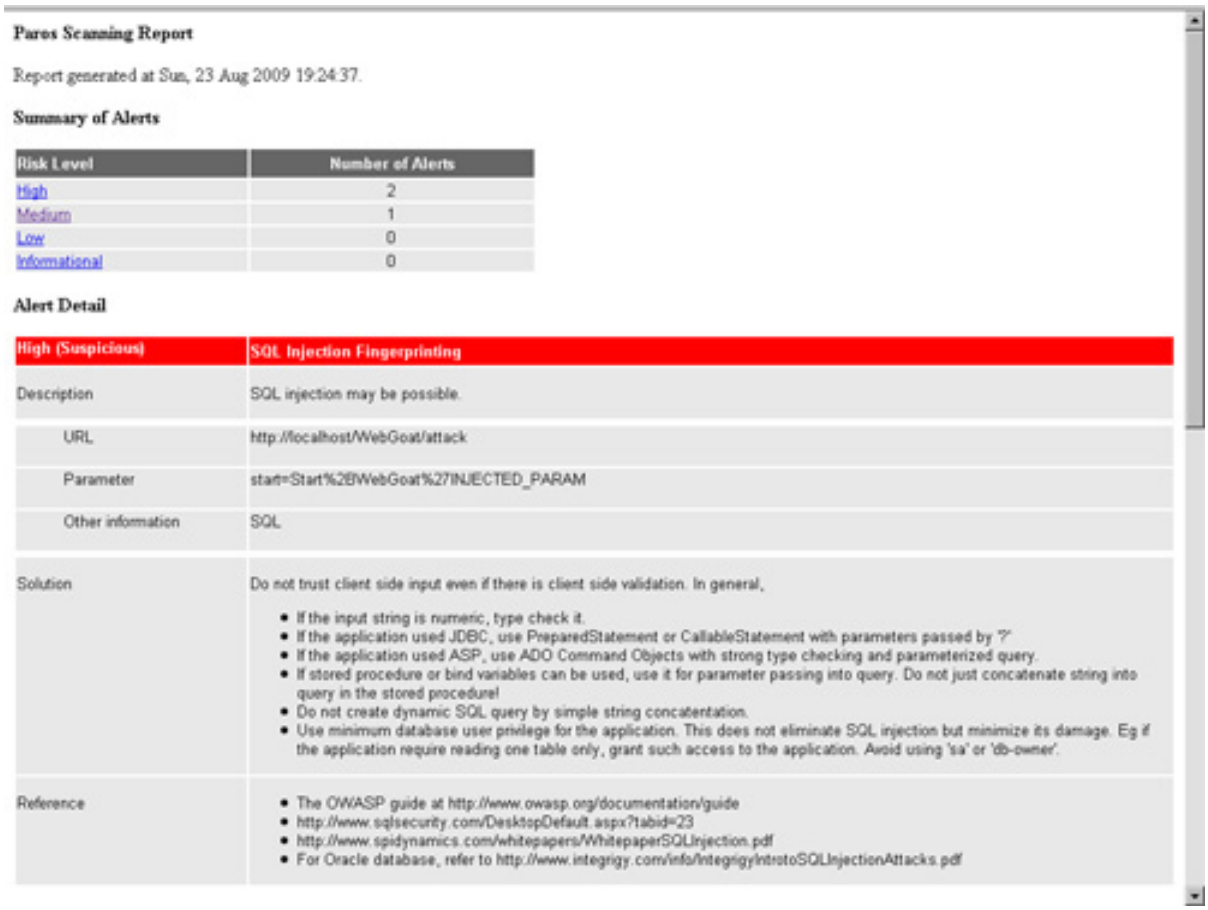
Figure 7. Viewing alerts



You can expand the alerts to show them in greater detail. You can also view the report, which offers much more information. First, however, save the scan results by clicking **File > Save As**, choosing a location and file name for your scan, and then clicking **Save**.

To view a report, click **Report > Last Scan Report**. Click **OK** in the pop-up window, and a new browser tab opens with the report of the scan results, as shown in Figure 8. This report provides a great deal of information regarding each vulnerability detected, including a reference to the OWASP page that deals with the specific exploit.

Figure 8. The Paros report



Paros Scanning Report

Report generated at Sun, 23 Aug 2009 19:24:37.

Summary of Alerts

Risk Level	Number of Alerts
High	2
Medium	1
Low	0
Informational	0

Alert Detail

High (Suspicious)	SQL Injection Fingerprinting
Description	SQL injection may be possible.
URL	http://localhost/WebGoat/attack
Parameter	start=Start%2BWebGoat%27INJECTED_PARAM
Other information	SQL
Solution	Do not trust client side input even if there is client side validation. In general, • If the input string is numeric, type check it. • If the application used JDBC, use PreparedStatement or CallableStatement with parameters passed by '?' • If the application used ASP, use ADO Command Objects with strong type checking and parameterized query. • If stored procedure or bind variables can be used, use it for parameter passing into query. Do not just concatenate string into query in the stored procedure! • Do not create dynamic SQL query by simple string concatentation. • Use minimum database user privilege for the application. This does not eliminate SQL injection but minimize its damage. Eg if the application require reading one table only, grant such access to the application. Avoid using 'sa' or 'db-owner'.
Reference	• The OWASP guide at http://www.owasp.org/documentation/guide • http://www.sqlsecurity.com/DesktopDefault.aspx?tabid=23 • http://www.spidynamics.com/whitepapers/WhitepaperSQLInjection.pdf • For Oracle database, refer to http://www.integrigy.com/info/IntegrigyIntrotoSQLInjectionAttacks.pdf

Testing for false positives

Although scanners are a good way to find possible vulnerabilities in a Web site, the best security companies always test possible vulnerabilities by hand for false positives. Testing for these possible exploits means actually going to the areas of your site that were reported as vulnerable, inserting SQL code or a script into the site itself to see what the response is, and then going into the site itself and plugging any exploits. Large firms often hire professional programmers who specialize in this type of testing (called *vulnerability validation*), but as a developer, you can perform some of these tests yourself. Not only will doing so help you secure your site a bit more, but it will help you as you build future sites.

False positives with SQL injection

Using the Hacme Casino site again, let's look at the vulnerability that WebScarab found: an SQL injection exploit at the login. With Hacme Casino running, enter the SQL code that WebScarab used successfully—`') OR 1=1--`—in the Login input area of the site. When you click **Login**, the account Andy_Aces is opened. Because 1=1 is going to be true, the most basic form of an SQL injection works here. As for

Andy, he has the bad luck of being the first account in the database.

Just how did this work? In the back end, the database runs a query like this:

```
SELECT * FROM users WHERE (username=username AND
password=password)
```

The bit of code injected through the **Login** box turned the valid query into this:

```
SELECT * FROM users WHERE (username=’’) OR 1=1—AND
password=’’)
```

This query returns a successful login for the first user of the site, which is unfortunately Andy.

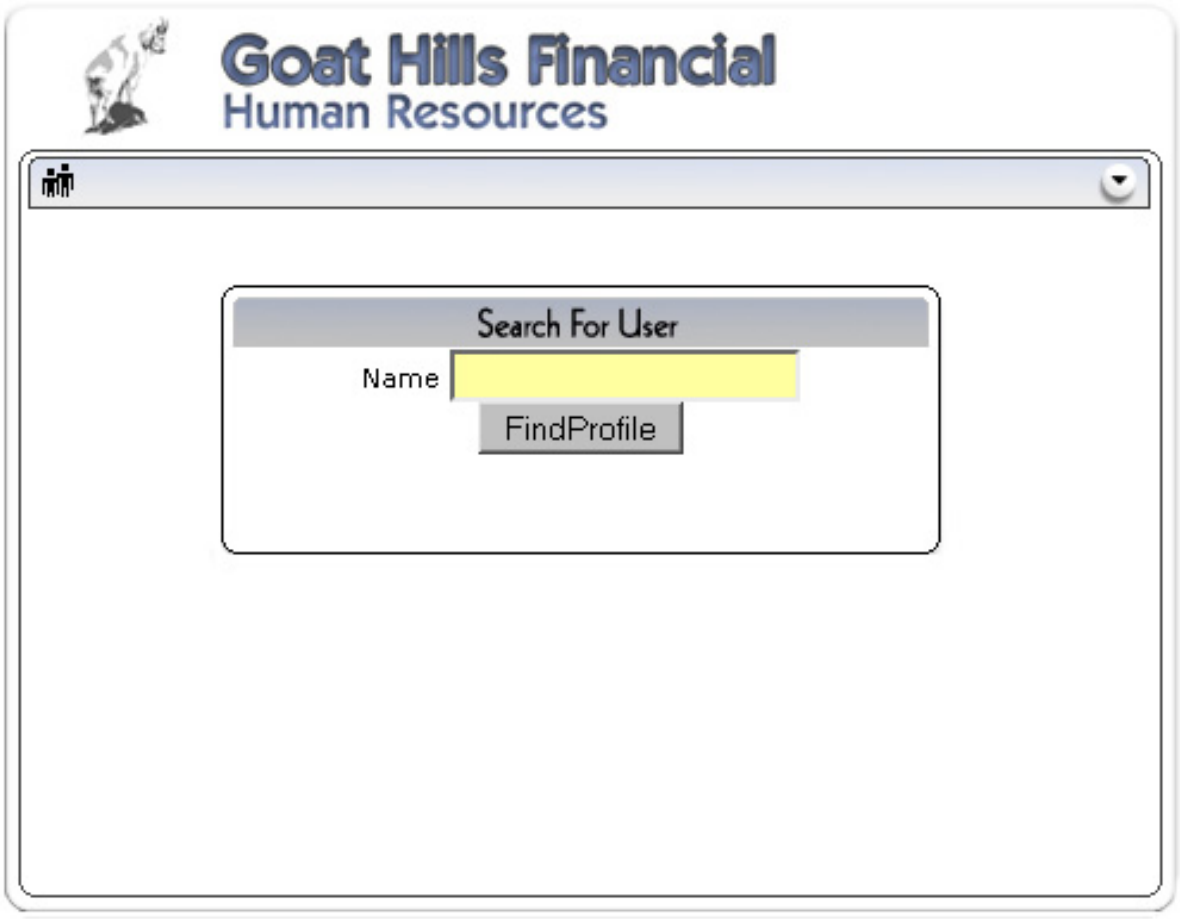
Of course, this example only scratches the surface of an SQL injection. As someone becomes more skilled in attacking Web sites, he or she can actually exploit this vulnerability to create users, change passwords, and extract sensitive data from the site. Protecting your site against these attacks involves taking a few steps:

- Use parameterized queries or stored procedures to access a database as opposed to using string concatenation. Parameterized queries require that you define all the SQL code and then pass in each parameter to the query later. This allows the database to distinguish between code and data, so it doesn't matter what type user input is supplied. Stored procedures are similar to parameterized queries in that they require you to define the SQL code first, then pass in the parameters later. These elements differ in that the SQL code for a stored procedure is defined and stored in the database itself, then called from the application.
- Sanitize user input by "whitelisting" the characters allowed. If you are asking for a name, only allow a-z and A-Z. For phone numbers, limit characters to 0-9. Use the appropriate validation technique supported by your database to do this.
- Escape user-supplied input using the character escaping scheme set up by your database. Escaping special characters tells your database that the characters provided in the query are in fact data, not code. If all user-supplied input is then escaped using the proper scheme, the database will not confuse that input with SQL code you have written.
- Don't give attackers any help. Make sure error messages don't give away information that can be later used against the site.

False positives with XSS

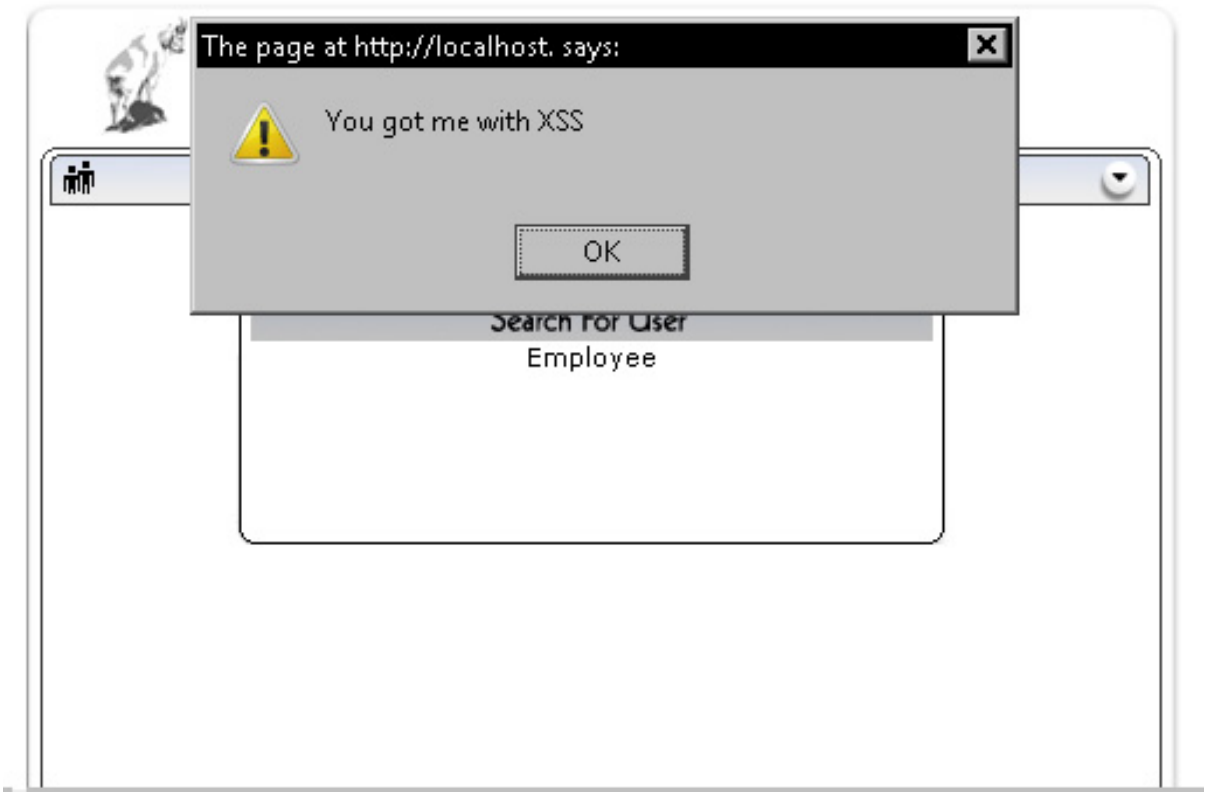
To demonstrate an example of an XSS attack, turn back to WebGoat. Start this site by clicking **Cross Site Scripting > LAB: Cross Site Scripting > Stage 1: Stored XSS**. Log in to the demo site—Goat Hills Financial—with a user name of `Larry Stooge` and the password `larry`. Now, an attacker would need to find somewhere to input a malicious script. The `Search Staff` function probably has a text box where someone can enter a name: Click **Search Staff** to reveal it, as shown in Figure 9.

Figure 9. An XSS entry point



In the **Name** box, type `alert("You got me with XSS");`. This script may only show an alert box like you see in Figure 10, but think about what could happen if the script could redirect a user to a fake human resources site where he or she had to enter a social security number, home address, bank information, and so on. Altering the action associated with the **OK** button can do this. For really creative malicious hackers, the sky is the limit if the user trusts them.

Figure 10. A successful XSS attack



To protect against XSS attacks, you need to scrub all inputs. If the input will later be used as parameters to operating system commands, scripts, and database queries, then it is essential that you do so. You can scrub user inputs by escaping untrusted data before allowing it to be inserted into HTML element content and HTML common attributes. OWASP recommends that all ASCII values less than 256 be escaped in the latter. JavaScript™ data values, HTML-style property values, and HTML value attributes can also be escaped. Of course, before you escape input data, make sure you have validated any input to your Web site. If you are looking for specific input values, numbers, letters, e-mail addresses, and the like, then only allow this type of data and refuse anything else.

Conclusion

In looking at some of the more common attacks that Web sites are subject to, it is important that you have an understanding of how these attacks work. By knowing what attackers are looking for and addressing these vulnerabilities, you can keep less experienced attackers from being able to breach a site and cause too much trouble for the more organized attacker to bother. Although the counter-measures listed in this article and the scans performed represent only the surface of how these attacks work and are prevented, they provide you with a foundation on which you can build.

Resources

Learn

- Prevent a cross-site scripting attack (Anand Sharma, developerWorks, February 2004): Discover ways to prevent XSS attacks on your site.
- Secure programmer: Validating input (David Wheeler, developerWorks, October 2003): Discover one of the first lines of defense in any secure program—validating input.
- Web application security statistics: The WASC presents an "industry wide effort to pool together sanitized website vulnerability data and to gain a better understanding about the web application vulnerability landscape."
- Website Security Statistics: See WhiteHat Security's sixth report on Web site security.
- developerWorks Web development zone: The Web development zone is packed with tools and information for Web 2.0 development.
- IBM technical events and webcasts: Stay current with developerWorks' technical events and webcasts.
- Check out My developerWorks: Find or create groups, blogs, and activities about Web development or anything else that interests you.

Get products and technologies

- Hacme Casino: Download Hacme Casino and try your hand at the skills learned in this article.
- WebScarab and WebGoat: Download the test sites from OWASP.
- Paros Proxy: Download the tool and scan your sites.
- IBM product evaluation versions: Download these versions today and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

About the author

Jeff Orloff
Jeff Orloff is a consultant for Sequoia Media Services, where he works to secure social media sites and Web applications for the companies deploying them. He also works with the School District of Palm Beach County as a technology coordinator. You can reach Jeff at jeff@sequoiamediaservices.com.