Perhaps one of the most interesting projects that is being kindly funded by Google's deep pockets is that of "Rockbox as an Application" or RaaA.

Increasingly, consumer electronics devices that are capable of music playback are shipping with very capable operating systems already, and it makes little sense to replace an entire operating system just to get superior music playback on such a device. Because there has been code in the Rockbox repository for a long time to build a "Rockbox Simulator", which runs on a PC using SDL, an opportunity to turn this into a real application that could execute on Android, Symbian, Maemo or even Windows Mobile (god forbid!) already existed, and now is being put into action. The aim of this project is to separate the playback engine from the firmware portion of the code, allowing you to install Rockbox on your phone without sacrificing its ability to make phone calls or surf the Web. It's thought that this is a better idea than trying to add a network stack and all the other goodies you'd need to make Rockbox a convincing alternative OS on your mobile phone. In actual fact, someone forked Rockbox's simulator a few years ago to the Motorola RockR series of phones in just this way, but sadly, the folks responsible for this port never gave the project any decent code back, and that particular set of targets remains unsupported in the trunk of the Rockbox codebase.

The majority of Rockbox's codebase is written in C, but there's a smattering of routines written in assembly code where it makes sense to squeeze the absolute last ounce of performance out of the player. Most of those routines are in the codecs, where every clock cycle counts, especially with some of the heavyweight formats like Monkey's Audio, which even the most powerful Rockbox players struggle with at maximum compression.

The Rockbox kernel has been written from the ground up, and among its features there are two things that probably will stand out as substantially different to people who are familiar with other modern OS kernels. First, because the vast majority of Rockbox-capable DAPs do not possess an MMU, Rockbox's kernel does not feature any method of dynamic memory allocation. This has been a contentious issue in the Rockbox community for a long time now, and new developers constantly bring it up. The lack of MMU is, of course, not a complete showstopper. It's perfectly possible to come up with a malloc implementation that does not require an MMU, and this is often the point they make. The reason the project has resisted it historically, and continues to do so to this day, is that any implementation requires a pool of free memory from which to allocate. The Rockbox old guard all argue that this is a waste of precious resources on the often extremely memory-constrained platforms on which Rockbox executes, and that every piece of available memory should be used to buffer as much music as possible. This means people coming to programming for Rockbox for the first time have to get used to thinking about statically allocating all the memory they need at compile time, a mode of programming foreign to most developers these days.

The second interesting feature in the Rockbox kernel is the threading model, which is cooperative rather than preemptive.

The second interesting feature in the Rockbox kernel is the threading model, which is cooperative rather than preemptive. This is mainly a historical artifact these days, and there has been repeated discussion about moving to a preemptive model, which so far has failed to reach a conclusion. In the meantime, this means, again, that anything designed to run on the Rockbox kernel has to have its execution thought about carefully and must consciously yield to other threads at opportune moments if it is to play nicely with the rest of the system.

As highlighted previously, Rockbox provides a rich plugin API that can be used via either compiled C code or in up-to-date versions via a growing library of LUA functions. A few support programs also go with Rockbox, not the least of which is the Rockbox Utility, a multiplatform installation tool. This is written in C++ using the Qt toolkit, and a version for all three major PC operating systems is maintained in statically linked form to make using it as easy as possible. There are some 92 people currently listed as having commit access to the central Subversion repository for Rockbox, with 40 of those active at the current moment. In addition to them, there is a sizable community of patch authors, artists and general helpers contributing to both the project's health and also to its vibrant and friendly atmosphere.

For those not skilled with a text editor and compiler, additions to and maintenance of the documentation are always welcome. Although it would be foolish to claim that the documentation is perfect, most of those involved in the project would say that as far as most open-source projects go, the documentation is pretty good. In addition to the manual and the wiki, which provide the static documentation for the project, there are numerous other ways for people to find and receive help. There are two active mailing lists, one aimed at the end user, which has proved to be perhaps the most popular support option for the community of blind users, and another where those who are interested in Rockbox development can exchange ideas. The Rockbox forums also are extremely active, and indeed, a lot of the Rockbox "staff" have made their entrance into the community by helping out others here. As with most other open-source projects, there is also an extremely active IRC channel on the Freenode network (**www.freenode.net**)—#rockbox is active for most of the day due to the community's diverse geographical (and some might say insomniacal!) nature. The channel is logged to make sure all discussions are searchable in case a developer misses something critical or otherwise interesting, and these logs are published on the project's Web site. Also, never let it be said that the Rockbox community is slow to adopt new technology. Recently @rockboxcommits has made its debut on Twitter for those who are keen to keep track of every change in the source code repository!■

Bryan Childs is a systems manager at a financial services software house in London. He likes cocktails and behaving in a disreputable manner whenever he's out of the office. He can throw a frozen egg over a ten-story building.

Web Application Security Testing with Samurai

Web site vulnerabilities often occur in very non-obvious ways. Whether you're a Web developer or run a Web site, you need to understand how it's done and how to test your site. JES FRASER

Almost every week the media picks up on another case of sensitive data being retrieved from Web sites with bad security. Web application security never has been more important, yet many Web sites never have been audited for security in a meaningful way. Although careful application architecture can help minimize security risks in code, a complete approach to Web application security considers the entire life cycle of the application, from development to deployment. In order to test whether your Web site is truly secure, consider using the same tools attackers do.

Penetration testing is the art of assessing the security of a system by simulating an attack or series of attacks. The goal of the penetration test is not necessarily to exploit the system if a flaw is found, but to audit potential attack vectors stringently and provide data that can be used to evaluate the potential risk of an exploit, and to find a solution to secure the system.

The Samurai Web Testing Framework is a security-oriented distribution that focuses on penetration testing for Web applications. It includes a variety of graphical, command-line and browser-based tools to test for common Web vulnerabilities. It's available as a live CD image from **samurai.inguardians.com**.

In this article, I look at using Samurai to test for a couple of the top Web application security risks as defined by the Open Web Application Security Project (OWASP). This is not specifically a list of attack vectors. Technically, many of the risks listed below are exploited using various forms of SQL injection. Rather, this list was developed by OWASP combining "threat agents, attack vectors, weaknesses, technical impacts and business impacts...to produce risks". OWASP's top ten risks (**www.owasp.org/index.php/Top_10**):

1. Injection flaws.
2. Cross-site scripting.
3. Broken authentication and session management.
4. Insecure direct object references.
5. Cross-site request forgery.
6. Security misconfiguration.
7. Insecure cryptographic storage.
8. Failure to restrict URL access.
9. Insufficient transport layer protection.
10. Unvalidated redirects and forwards.

In the interests of keeping the scope of this article manageable, I focus on injection flaws and cross-site scripting.

Disclaimer: please do not try any of these examples on production Web sites. *Linux Journal* recommends that you set up a virtual environment with a copy of your Web site to test for vulnerabilities. Do not test over the Internet. Never use any of these examples on a Web site that is not yours. *Linux Journal* is not responsible for any damage to data or outages to services that may arise from following any of these examples.

Injection Flaws

Injection flaws occur when the application passes user input to an interpreter without checking it for possible malicious effects. Injection flaws can include operating system command injection, LDAP injection and injection of many other interpreters called by a Web application using dynamic queries. One of the most common injection vectors is SQL injection. Depending on the specific vulnerability, attackers could read passwords or credit-card numbers, insert data into the database that gives them access to the application or maliciously tamper with or delete data. In extreme cases, operating system files could be read or arbitrary system commands could be executed—meaning game over for the Web server.

Login forms are primary targets for SQL injection, as a successful exploit will give attackers access to the application. To start testing an application for SQL injection vulnerabilities, let's use some characters that have special meaning in SQL to try to generate an SQL error. The simplest test, using a single quote (') as the user name, failed to generate an error, so let's try a double quote followed by a single quote ("'):

```
SQL Error: You have an error in your SQL syntax; check the manual
           that corresponds to your MySQL server version for the
           right syntax to use near '""'' at line 1

SQL Statement: SELECT * FROM accounts
               WHERE username='""' AND password='""'
```

Not only is this form vulnerable to SQL injection, but also the

error message has thrown up the exact SQL statement being used. This is all the information you need to break into this application, by using the following text in both the user name and password fields:

```
' or 1=1 --
```

This changes the original query to one that can match either the correct user name and password, or to test if 1=1. Because 1=1 will evaluate to true, the application accepts this as your login credentials and authenticates you. Even worse, assuming you're not attackers, once you're logged in, you now can see that you're the admin user. This is because the SQL query will look at each row, one by one, to see on which row the query returns true. Because you've tampered with it always to return true, MySQL will return the first row. Because the first user created is quite often the administrator or root user (Figure 1).

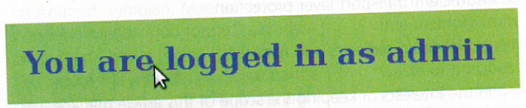


Figure 1. The first row of the users table often contains the administrator user.

Not all applications are going to prove so easy to break into—particularly those that do not divulge as much information about the database and table structure in the error message. An auditing tool will let you iterate over a range of possible strings quickly. w3af, the Web Application Attack and Audit Framework, has a range of plugins to assist in scanning for and exploiting vulnerabilities, including SQL injection.

Launch the w3af GUI from the Applications→Samurai→Discovery menu. Enter the Web site URL you'd like to test in the Target: field, and then expand the options under discovery in the plugin box. Scroll down until you find the webSpider plugin, and check its box to enable it. In the pane to the right, the options for the webSpider plugin will be shown. Tick onlyForward, and select Save Configuration. Now, scroll back to the top of the plugins box, and expand Audit. Scroll down to sqli and check its box to enable it.

Once the scan is completed, you can look at the Results tab and see that w3af found seven separate places within the application that could be exploited by SQL injection (Figure 2).

Another tool included with Samurai that can discover SQL injection vulnerabilities is Grendel-Scan. Launch Grendel-Scan from the Applications→Samurai→Discovery menu. Under Base URLs, insert the URL of the Web application you would like to test and click Add. Untick Enable Internal Proxy. Let's use Grendel's Web spidering module instead. Under Scan Output, select a directory for Grendel-Scan to store its output report. This directory must not exist. The application will expect to create it. Select Start Scan from the Scan menu to start.

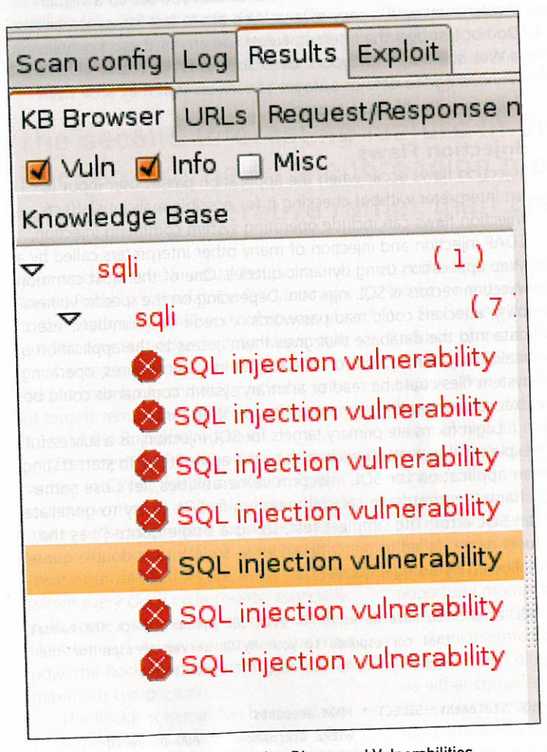


Figure 2. w3af Results Tab, Showing Discovered Vulnerabilities

The scan will take some minutes to complete, depending on the size of the Web application. Once finished, navigate to the designated output directory to view the report. Here, report.html tells you that among other issues, a possible SQL injection vulnerability was found (Figure 3). The clean output of Grendel-Scan makes it a great tool to send attractive and easy-to-read vulnerability reports to upper management as part of a reporting requirement.

Possible SQL Injection	
Severity:	High
URL:	See description
Description:	When a single quote (') was appended to the parameters listed below, a SQL error message was returned. This could indicate a SQL injection vulnerability.

Figure 3. A Section of Output from Grendel-Scan

Cross-Site Scripting (XSS)

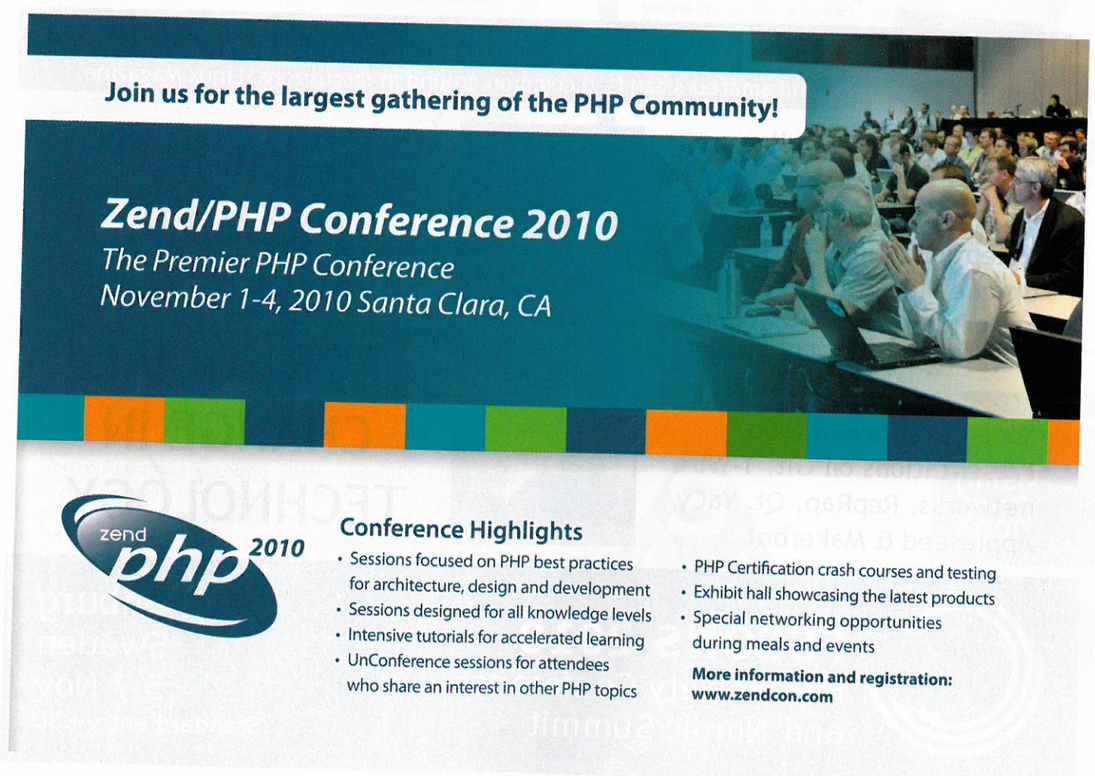
Cross-site scripting can occur when the application accepts user input and sends it back to the browser without validating it for malicious code. There are two types of XSS attacks: reflected and stored. A stored attack injects code into the page that is persistent and will be activated by the victim's browser when requesting the page. A common example of a stored cross-site scripting attack would be a script that is injected through a comment field, forum post or guestbook. If the application doesn't guard against potentially malicious input data and allows the script to be stored, the

Statement of Ownership, Management, and Circulation

1. Publication Title: *Linux Journal*
2. Publication Number: 1075-3583
3. Filing Date: September 8, 2010
4. Issue Frequency: Monthly
5. Number of Issues Published Annually: 12
6. Annual Subscription Price: $29.50
7. Complete Mailing Address of Known Office of Publication: 2121 Sage Road, Ste 310 Houston, TX 77056
   Contact Person: Mark Irgang 713-344-1956 x111
8. Complete Mailing Address of Headquarters of General Business Office of Publisher: 2121 Sage Road, Ste 310 Houston, TX 77056
9. Full Names and Complete Addresses of Publisher, Editor, and Managing Editor:
   Publisher: Carlie Fairchild PO Box 980985 Houston, TX 77098
   Editor: Doc Searls PO Box 980985 Houston, TX 77098
   Managing Editor: Jill Franklin PO Box 980985 Houston, TX 77098
10. Owner(s): Carlie Fairchild PO Box 980985 Houston, TX 77098; Joyce Searls PO Box 980985 Houston, TX 77098; Adele Soffa PO Box 980985 Houston, TX 77098
11. Known Bondholders, Mortagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortages, or Other Securities: None
12. Tax Status: Has not Changed During Preceding 12 Months
13. Publication Title: *Linux Journal*
14. Issue Date: October 2010

15. Extent and Nature of Circulation	Average No. Copies Each Issue During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
a. Total Number of Copies: (Net press run)	45,361	40,650
b. Paid and/or Requsted Circulation		
(1) Paid/Requested Outside-County Mail Subscriptions on Form 3541.	15,434	14,633
(2) Paid In-County Subscriptions Stated on Form 3541	0	0
(3) Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Non-USPS Paid Distribution	12,990	12,368
c. Total Paid and/or Requested Circulation	28,424	27,001
d. Free Distribution Outside the Mail		
(1) Outside-County as Stated on Form 3541	434	309
(2) In-County as Stated on Form 3541	0	0
(3) Other Classes Mailed Through the USPS	0	0
(4) Free Distribution Outside the Mail	3,833	2,555
e. Total Free Distribution	4,267	2,864
f. Total Distribution	32,691	29,865
g. Copies Not Distributed	12,670	10,785
h. Total	45,361	40,650
i. Percent Paid and/or Requested Circulation	86.9%	90.4%

PS Form 3526



This powerhouse tool is capable of scanning for and exploiting almost any vulnerability you care to name.

browser of the next person to load the page will execute the malicious code.

By contrast, a reflected or nonpersistent cross-site scripting attack requires the victim to click on a link from another source. Here, instead of being stored in the Web application's database, the malicious script is encoded in the URL. XSS exploits can allow attackers to deface a Web site, redirect legitimate traffic to their own site or a site of their choice, and even steal session data through cookies.

This first test utilizes a formidable tool that enjoys wide popularity among attackers—a Web browser. Although automated tools can take the pain out of iterating over a long list of tests, the simplest method to start testing your site for XSS vulnerabilities is by using a browser. You'll need to find a page on the application you'd like to test that accepts user input and then displays it—something like a guestbook, comment field or even a search dialog that displays the search string as part of the results.

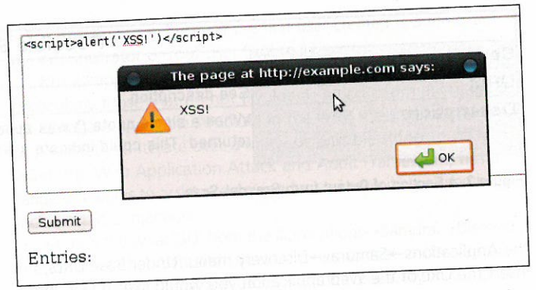


Figure 4. XSS Vulnerability Executed

The canonical check for an XSS vulnerability is considered to be the snippet of code below:

```
<script>alert('xss!')</script>
```

If the Web site isn't properly filtering input, this script will execute in the browser and then display a pop-up message with the text "xss". There's some bad news for my little Web application here. It's so vulnerable that this simplest of strings was able to exploit it (Figure 4).

"brilliant...free software, hacktivism, and social practices acting together and demonstrating that IT experts, users, and amateurs can find common ground in discussions" Linux Magazine

- 65 Speakers in 56 talks
- Premier event for Free Culture and Free Software
- Exceptional university venue
- Unique platform for brainstorming & discussion
- Workshops hosted by leading projects & organisations
- Social events and food
- Presentations on Git, 1-Wire networks, RepRap, Qt, YaCy, Appleseed & Makerbot

THE WORLD'S PREMIER CONFERENCE FOR SOCIAL CHANGE IN TECHNOLOGY

FSCONS 2010
Free Society Conference and Nordic Summit

Gothenburg
Sweden
5-7 Nov
Standard entry €30

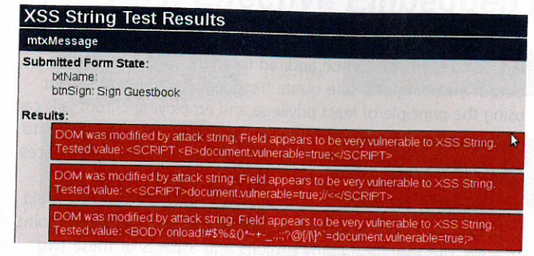


Figure 5. XSS Me Test Results

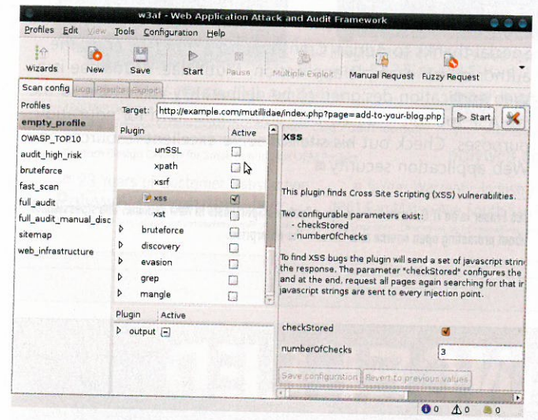


Figure 6. w3af Configuration

Usually, at least some form of input filtering is employed, and this string won't get through intact. By examining the source of the generated page, you can try to diagnose what characters are being filtered to craft more sophisticated strings. If you inject the same string into a slightly more secure Web form, you can see by the result that the script tags are being stripped out and the single quotes are escaped:

```
Name: Me!
Message: alert(\'XSS!\')
```

You could continue encoding different parts of the string and observing the results to try to get past the filter. Or, to speed up the process, you could use an automated tool. Samurai includes the XSS Me plugin for Firefox by Security Compass. Navigate to the page you'd like to test, then select XSS Me→Open XSS Me Sidebar from the Tools menu. Select Test all forms with top attacks from the sidebar. For the curious, the strings that the XSS Me uses can be viewed in Tools→XSS Me→Options→XSS Strings, and more strings can be added.

Figure 5 shows that XSS Me has discovered that this application is vulnerable to DOM-based XSS attacks. This is a particularly

ASA COMPUTERS

Want your business to be productive?
The ASA servers powered by the Intel® Xeon® Processor provide the quality and dependability to keep up with your growing business.

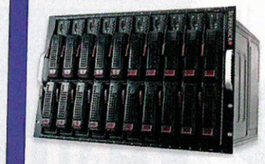
" Since 1989 Integration and service with pride "

ASA 1U Series
Intel® Atom™ processors D510 and D410 based platform and are designed for embedded industrial PC (IPC) applications. These quiet, energy saving solutions make ideal network appliances, print and email servers.

ASA 2U Series
QPI, Intel® Xeon® Processor 5500 Series in a high-density 2U form-factor are ideal for network infrastructure, front-end enterprise, and minimal downtime cluster server systems maximum upto 4 nodes in 2U.

ASA 3U Series
Server series excel under ISCSI/NAS/JBOD environments, the 3U servers support high availability storage and mission critical business applications.

ASA 4U Series
Optimized for enterprise-level high-capacity storage applications, features 36x (24 front + 12 rear) 3.5" Hot-swap HDD bays, reliable and hassle-free maintenance storage system.



ASA Blade Series
Blade server with, Highest computing density (20 DP nodes and 2.56TB mempry in 7U) Fastest and Most Cost-Effective Networking Solution (Infiniband DDR/QDR support)

E-mail - sales@asacomputers.com
Call - 1800-REAL-PCS

ASA Computers, Inc
645 National Ave,
Mountain View. CA 94043
www.asacomputers.com

intel Xeon inside
Powerful. Intelligent.

Intel, the Intel logo, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

insidious way of exploiting an application, as it doesn't rely on the application embedding the malicious code in the HTML output.

Another tool that can scan for XSS vulnerabilities in Samurai is, again, w3af. This powerhouse tool is capable of scanning for and exploiting almost any vulnerability you care to name. Here, let's configure it to scan the Web application for XSS vulnerabilities.

Launch the w3af GUI from the Applications→Samurai→Discovery menu. Enter the Web site URL you'd like to test in the Target: field, and then expand the options under discovery in the plugin box. Scroll down until you find the webSpider plugin and check its box to enable it. In the pane to the right, the options for the webSpider plugin will be shown. Tick onlyForward and select Save Configuration. Now, scroll back to the top of the plugins box, and expand Audit. Scroll down to xss, and check its box to enable it (Figure 6).

Click Start to start the audit process. Depending on the size of the Web application, this could take a few minutes as the spider plugin discovers every URL, and then the audit plugin tests all applicable forms for XSS vulnerabilities. Once complete, navigate to the Results tab, which will itemize the vulnerabilities found.

Conclusion

Although useful, penetration testing is only part of the picture. To truly address these risks, applications must be designed, implemented and deployed with security in mind. Code analysis tools are helpful in locating places where application code deals with user inputs, so the code can be audited for input validation, strong output encoding and safe quote handling. Careful deployment using the principle of least privilege and employing chroot jails can help minimize the damage attackers can do if they gain access to your application. Never allow your database or Web server process to run as the root user.

Hopefully, this article can serve as a jumping off point to help you approach Web application security from a more active point of view, but there are many exploits and aspects of these two specific exploits that aren't covered here. For further reading, see the OWASP Wiki at **www.owasp.org**.

Acknowledgement

Special thanks to Adrian Crenshaw from irongeek.com, the author of our example application, Mutillidae. Mutillidae is a Web application designed to be deliberately vulnerable to the OWASP top ten in an easy-to-understand form for education purposes. Check out his site for some excellent resources on Web application security.■

Jes Fraser is an IT Consultant from Open Systems Specialists in New Zealand. She's passionate about promoting open source and Linux in the enterprise.