Security in Open Source Web Content Management Systems

Users of Web content management systems (WCMSs) lack expert knowledge of the technology itself, let alone its security issues. Complicating this, WCMS vulnerabilities are attractive targets for attackers, leaving the applications and their nonexpert users open to exploitation.

Michael Meike
Trusted Bytes

Johannes Sametinger and Andreas Wiesauer
Johannes Kepler University

Many Web applications, such as those for e-commerce or collaboration, use out-of-the-box Web content management systems. WCMSs let users who don't have in-depth development knowledge easily build a customized Web site with broad functionality. For small- and medium-sized enterprises, open source WCMSs offer an easy to use, low-cost alternative to commercial software. However, these systems raise significant security issues.

Security is critical to any Internet-connected information system; vulnerabilities can create serious consequences for system users and operators, including stolen credit-card data or customer information. Because of their wide usage, open source WCMSs are a desirable target for attackers. Once malicious users discover a vulnerability in a particular WCMS, they can carry out attacks on many—if not all—of the applications built with it. Open source WCMS developers are aware of this and have established security teams, Internet forums, and security tips for users. Still, the security of such systems remains unclear. To shed light on it, we've selected two popular open source WCMSs based on PHP: Hypertext-Preprocessor (PHP) and analyzed them for well-known vulnerabilities.

Web Content Management Systems

Heidi Collins defines content management as maintaining, organizing, and searching across information sources, both structured (databases) and unstructured (documents, emails, video files, and so on).[1]

Among CMSs, we distinguish between those focused on the Web and the enterprise. Historically, the content management concept originated from organizations' efforts to manage Web content.[2] The enterprise content management (ECM) concept reaches beyond Web content management, however, addressing management of "the convergence of all front-end applications and devices with back-end document/file management systems and databases."[2] ECM involves not only technical systems, but also "the strategies, tools, processes and skills an organization needs to manage its information assets over their lifecycle."[3]

In contrast, according to current understanding, a WCMS supports creating and publishing content structured in Web formats, such as HTML, XHTML, XML, and PDF. A WCMS also lets users review, approve, and archive content, and (sometimes) offers version control.[4] Using such functions, users can implement an editorial process that comprises several roles with varying privileges, including authors, reviewers, and consumers.

An organization might, for example, use a WCMS to build corporate Web sites, online shops, or community portals. The major advantage of a WCMS is that it lets site creators modify content without having to edit code or possess other specialized knowledge. Organizations typically store the content in databases and publish, modify, and remove it using graphical user interfaces.

A WCMS can be divided by front- and back-end functionality. The front end presents available content to consumers, who typically don't have permission to change or edit the content. However, front-end users are sometimes granted special permissions, such as to submit comments on articles.

Publishing and editing is done via the application's back end, which is the workplace for authors and reviewers. Authors typically enter their content using a rich text editor, which creates HTML, XHTML, or XML markup. They render this markup by applying style sheets—such as CSS or XSL—which authors can adapt to suit their design needs.

WCMS Security

In software systems, security vulnerabilities are defects at either the design or implementation level. Gary McGraw defines a *bug* as an implementation-level software problem and a *flaw* as a problem that's "certainly instantiated in software code, but is also present on the design level."[5] Examples of flaws include error-handling problems and broken or illogical access control. Bugs and flaws create *risks*, which are the probability that a flaw or a bug will impact the software's purpose: risk = probability × impact.[5] Software defects might exist for a long time before attackers actually exploit them.

Networking applications—especially those exposed to the Internet—are much more vulnerable to threats than conventional stand-alone desktop applications as many more users can access them, and access is much harder to control. Often, attacks over the Internet are almost fully automated, and many tools let people with even minor technical knowledge (known as "script kiddies") exploit vulnerabilities.

Key Vulnerabilities

As a Web application, a WCMS is an attractive target for attackers and a major source of security vulnerabilities.[6] Threats affect one or more security aspects. According to several experts,[7,8] Web application threats include

- *Data manipulation*. This type of attack violates data integrity, and the resulting data loss or perversion can have serious consequences. Common attack techniques here include parameter manipulation and SQL injection.
- *Accessing confidential data*. Here, attackers access off-the-record data using techniques such as structured Query Language (SQL) injection and cross-site scripting (XSS).
- *Phishing*. This attack gathers confidential data—such as bank account information, social security numbers, or passwords—by contacting users under false pretences via email and luring them to Web sites where they're encouraged to enter personal data. For example, attackers might use XSS to gather user data by placing manipulated input forms on pages managed by a WCMS. They can also exploit WCMS vulnerabilities to lure users to replicated Web sites that mimic an official site, such as a bank, and gather data accordingly.
- *Code execution*. Attackers can exploit WCMS vulnerabilities to load files or programs containing defective code onto a Web server. They can do this by using even simple graphic files. The WCMS must carefully verify inputs because such an attack can harm not only the WCMS itself, but also other applications on the same server.
- *Spam*. Here, Web crawlers scan the Internet for valid email addresses and send spam accordingly. Attackers can also use an application vulnerability to send spam through the application's server, turning it into a spam relay server.

As these examples show—and Michael Howard and David LeBlanc note—Web applications in general, and WCMSs in particular, operate in a hostile environment.[9]

Attacks and Countermeasures

Attackers use various attack patterns, which are blueprints for creating a particular type of attack. Each attack consists of several phases of discovery and exploitation; the pattern generalizes the steps so that other malicious users can successfully attack the application. Attack patterns can include many dimensions, such as timing, resources required, and techniques.[10]

Because of a WCMS's wide application area, the possible harm due to attacks is manifold. If a WCMS is used as an e-commerce site, attackers might obtain confidential customer data, such as credit-card information. If the site's owner had failed to properly secure the WCMS, an attack disclosure could lead to claims for damages, as well as a general loss of customer confidence.

In other cases, attackers might gather user information, such as addresses and profiles, and sell the information to the site's competitors. A WCMS can also be sabotaged and rendered inaccessible. This could, for example, lead to a decline in sales.

On corporate WCMS Web sites, attackers might utilize security leaks to upload malicious code and harm a company's IT infrastructure. Attackers might also alter the company's Web site content by, for example, adding dubious and suspect content to tarnish the company's reputation.

To be considered secure, a Web application must ensure

- authentication, by verifying that entities or people are who they pretend to be;
- confidentiality, by hiding information from unau-

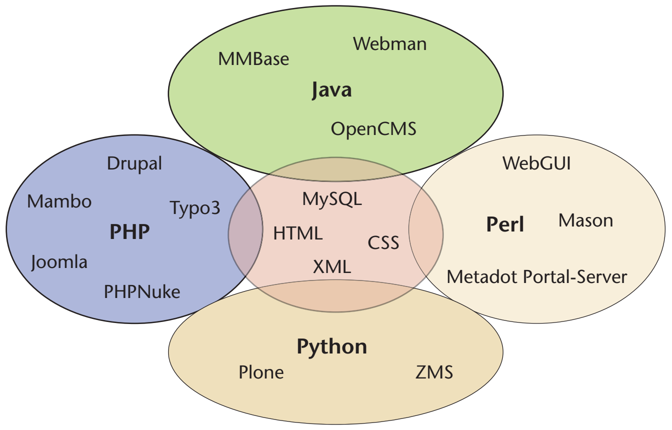

Figure 1. Open source Web content management systems and related technologies. All systems, regardless of implementation, use Web standards, such as HTML and CSS, and relational databases.

thorized people;

- integrity, by preventing unauthorized people from modifying, withholding, and deleting information; and
- availability, by performing operations according to their purpose over time.[11]

To achieve these goals, application developers can use several mechanisms, including sophisticated authentication, user access control, and mechanisms that determine when to maintain data confidentiality, such as when not to show credit-card numbers when verifying a person's financial state.

Security Analysis: Open Source WCMS

As the sidebar, "Open vs. Closed Source Security" notes, an open source system's primary attractions give rise to its vulnerabilities: its low cost and source code availability make it readily accessible to attackers seeking to locate and exploit application weaknesses. Still, open source WCMSs are widely available and widely used.

Figure 1 shows an overview of technologies and corresponding WCMSs. Most systems are developed in PHP, Java, Perl, or Python. As the ellipses indicate, all systems—independent of their implementation—use Web standards such as HTML, XML, and CSS and relational databases such as MySQL.

It's difficult to determine the exact number of sites powered by a specific WCMS. Various sources try to estimate such numbers using different metrics.[12] For our case study, we chose Joomla (www.joomla.org) and Drupal (www.drupal.org)—two open source systems that create complex Web content. These are among the most widely used systems according to such estimates. Additionally, UK Linux World named Joomla the Best Linux/Open Source Project in 2006 (www.joomla.org/announcements/general-news/2165-joomla-wins-again-at-uk-linuxworld.html). That same year, Joomla also won Packt Publishing's Open Source CMS Award, and Drupal took second place (see www.packtpub.com/article/open-source-content-management-system-award-winner-announced).

Joomla and Drupal have many similarities. They both use LAMP—that is, Linux, Apache, MySQL, and PHP. They both also use XML files to store configuration parameters and CSS for design and layout, and have similar functionality. Nonetheless, Joomla and Drupal have key differences that make them interesting from a security-comparison perspective, including different architectures and implementations.

Joomla

Joomla is a derivative of Mambo, a popular PHP-based WCMS, and has been used to build roughly 5 million Web sites worldwide. The system emphasizes ease of use, so even nontechnical users can create, edit, and maintain content. Joomla also offers many out-of-the-box Web site components, including forums and chat components, calendars, and blogging software. Joomla is easily customized for special needs and is in use on everything from private and small business Web sites to corporate portals.

Joomla developers are organized in a core team that's responsible for overall project management, and several working groups that handle particular issues, such as development, documentation, or translation. Joomla developers estimate that there are more than 140,000 active registered users on the Official Joomla community forum (http://demo.joomla.org/1.5/more-about-joomla/30-the-community/21-joomla-facts.html).

The Joomla forum comprises a security section in which users discuss security issues and submit possible vulnerabilities (http://forum.joomla.org/viewforum.php?f=432). This section also contains guidelines, tutorials, and hints for increasing security and mitigating risks. Security issues are categorized according to level (low, medium, or high) and are fixed in minor releases or by patches.

Drupal

Dutch students developed Drupal in what was originally an effort to implement a collaboration platform. Today, Drupal is used for numerous private and university Web sites, as well as for collaboration portals and e-commerce sites. DrupalSites.net, for example, lists several thousand Web sites powered by Drupal (www.drupalsites.net). Like Joomla, Drupal offers many additional modules, including newsletters and podcasting components.

Drupal developers have their own security team (http://drupal.org/security-team) that's responsible for accepting and evaluating security-related warnings, searching for vulnerabilities in the core application, and supporting module developers in fulfilling security requirements.[13] In recognition of security's importance, the Drupal Web site has a dedicated section to inform users about current vulnerabilities and appropriate patches.

Analysis Overview

For our analysis, we used Drupal 5.2 and Joomla 1.0.13. As of this writing, newer versions of both systems are available. However, our goal hasn't been to list a specific version's vulnerabilities, but rather to give a sense of the systems' security status and whether users can trust this security without further ado. Hopefully, both organizations will constantly release newer versions that fix known security problems. Indeed, updates to both applications fix some of the vulnerabilities we now describe, further confirming our analysis results. However, new versions can also introduce new problems, and attackers might find additional security holes.

We carried out our security analysis in several steps. First, we installed both systems and evaluated how different configuration settings—such as deactivating the PHP `safe_mode` setting—might influence security issues. Second, we performed simple penetration tests by sending various malicious input. We were aided here by several simple tools—including WebScarab (www.owasp.org/index.php/OWASP_WebScarab_Project) and TamperData (http://tamperdata.mozdev.org)—that let us perform simple security tests by manipulating parameters sent to Web servers, such as modifying data in HTTP request headers. As we describe later, we sent simple requests that could lead to XSS or SQL injection.

In our third step, we inspected the source code files of both Joomla and Drupal for additional problem areas. Both systems had almost 2,000 source files, which we searched for security-related strings. For example, `$_SERVER` indicates an access to the Web server's global variables, which might contain input from the Web client. The variables might also include hazardous input from malicious users. So, for our analysis, we simply reviewed the code to see whether the developers had taken appropriate measures before they used the variables' content. We also used the source files to understand and evaluate the systems' general security mechanisms, such as authentication or user session management.

In step 4, we used the knowledge gained in step 3 to send additional and more focused malicious requests. Finally, we evaluated community support for security issues by simply browsing the Joomla and Drupal Web sites and by checking for specific community activities, including security forums, FAQs, and checklists.

Finally, although it's possible to assess WCMS security using special penetration testing tools, we refrained from using commercial tools. However, we might consider using license-free tools—such as those for source code analysis—in future evaluations.

Open vs. Closed Source Security

Open source software's main advantage is its low cost—it's freely available and requires no licensing fees. In addition, because of the source code availability, users can tailor open source software to their specific needs. However, potential attackers can also use the source code to identify vulnerabilities.

Experts have differing opinions as to whether open source applications are more prone to security failures than commercial products; they also disagree on the overall quality of open source versus commercially developed applications.[1] Some experts argue that open source developers are less trained and therefore produce more failures and weaker designs than their commercial counterparts. Others assert that open source's "many eyeballs phenomenon"[2]—that is, that a whole community is involved in programming, testing, using, and offering feedback on open source applications—leads to quicker discovery and repair of failures.

Obviously, closed source software also suffers from security vulnerabilities. As John Viega and Gary McGraw noted, it's "a false belief that code compiled into binaries remains secret just because the source is not published." That is, when code is running, hackers have various methods for examining it—"security by obscurity" isn't as effective as many people think.[2]

References

1. G. Lawton, "Open Source Security: Opportunity or Oxymoron?" *Computer*, vol. 35, no. 3, 2002, pp. 18–21.
2. J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison-Wesley, 2002.

Analysis Results

We evaluated the two systems according to specific security categories and criteria. Table 1 offers a summary of the results.

Community

On the Internet, change happens quickly; knowledge about vulnerabilities also spreads quickly. It's thus necessary to rapidly respond to vulnerabilities and provide patches to prohibit their exploitation once they're publicly known. Development communities must be dedicated to providing both a secure system and information for their users on how to further ensure system security. Communities should also define processes for reporting vulnerabilities and tracking their status.

Both the Joomla and Drupal communities are

Table 1. Security analysis results.

	JOOMLA	DRUPAL
COMMUNITY		
Security patches	●	●
Vulnerability reporting	●	●
Hints on countermeasures	●	●
INSTALLATION		
Security hints	●	❍
Security settings	❍	❍
PARAMETER MANIPULATION		
HTTP header data	❍	❍
Super global arrays	●	●
Cookies	●	●
Remote Command Execution	●	●
Forms	◗	❍
CROSS-SITE SCRIPTING (XSS)		
APIs against XSS	●	●
XSS via URL parameter	●	●
XSS via search fields	●	●
XSS in other forms	●	●
XSS in back end	●	●
SQL INJECTION		
Any countermeasures	●	●
USER ADMINISTRATION		
Login: XSS or SQL injection	●	●
Secure passwords	❍	◗
Sessions at the server	❍	◗
Sessions at the client	●	●
Session hijacking	◗	❍
Access to functions	◗	●
SPAM		
Contact forms	●	◗
Spam relays	●	●
Email addresses	●	❍
MALICIOUS FILE UPLOAD		
Checking file endings	●	●
Checking file contents	❍	❍
ELEVATION OF PRIVILEGE		
Privileged users	❍	◗
Administrators	❍	❍
OPTIONAL MODULES		
Warnings	●	●
Security measures in core	❍	❍

●: Security requirement fulfilled

◗: Security requirement partially fulfilled; potential security risk

❍: Security requirement isn't fulfilled; definite security risk

dedicated to fulfilling these security requirements as follows:

- *Security patches.* Joomla provides specific security Web pages and newsgroups. Development team members occasionally reorganize the Joomla structure and engage new teams to improve system quality. Drupal has a separate security team. Both systems provide new versions periodically and urge their users to update to the current version. This is important as attackers can use a system's version history to find information about eliminated vulnerabilities and then search the Web for older versions to exploit.
- *Vulnerability reporting.* Users can report vulnerabilities on both systems' Web sites. Joomla security issues are discussed at the official forum, whereas Drupal provides a section with detailed security announcements and information on how to fix issues quickly.
- *Tips on countermeasures.* Joomla users discuss countermeasures at their forum, whereas Drupal team members publish information in the site's security section.

Both systems' communities pay adequate attention to security aspects, systematically tracking vulnerabilities and providing patches with security fixes.

Installation

Because many WCMS users are nonexperts, it's important that the installation process be as automated as possible. WCMSs have many configuration settings that might open or close specific vulnerabilities; if users choose the default settings, the system should be secure:

- *Security hints.* Joomla provides a pre-installation check and warns users of suboptimal security settings. Drupal has a simpler installation process that doesn't provide security issue hints.
- *Security settings.* Once installation is complete, neither system offers sufficient support for modifying the security settings. However, the settings' primary focus is to guarantee a trouble-free collaboration with third-party modules.

It's important that users are alerted to security issues during installation and that they can easily modify security settings thereafter. Here, both systems have ample space for improvements.

Parameter Manipulation

Parameter manipulation includes the ability to alter super global variables, cookie poisoning, remote command execution, and Web form data manipulation as follows:

- *HTTP header data.* If an Internet application doesn't check for valid HTTP header data, manipulated data can result in multiple answers to a single request. Thus, a proxy or cache server might send manipulated answers to clients.
- *Super global variables.* In PHP applications, super global variables contain information set by the Web

server or otherwise directly related to the execution environment. Attackers can manipulate these variables to carry out SQL injection or XSS. Therefore, the system must check these variables before processing them.
- *Cookie poisoning*. Cookies typically store user-related data—such as shopping cart items—on a user's local computer. However, attackers can poison cookies by altering their contents during transmission to the server. Therefore, a WCMS must not blindly trust cookie data.
- *Remote command execution*. Attackers can execute remote commands on the WCMS by including malicious PHP scripts stored on the computers they're operating. To achieve this, attackers must manipulate parameters such that the PHP operations are called with the malicious script's URL. Thus, the system must check parameters if their content can lead to the execution of operations such as `include()` or `require()`.
- *Web form data*. Web forms contain various ways of transferring data from the user to a server. It's rather easy to manipulate this data and send dangerous contents to a server.

Although Joomla and Drupal are quite prepared for parameter manipulation, they're not without deficiencies: neither system sufficiently filters HTTP headers and Web form data. Drupal, for example, stores all transmitted data unfiltered in an underlying database table. Consequently, the security system checks only the database query results.

Cross-Site Scripting

Attackers can manipulate Web servers' input data to contain script code, such as JavaScript. If the server sends this input to other clients without appropriate checks, the client side executes the script code. Thus, XSS isn't dangerous to the server itself, but to its clients. Take, for example, the input `<script>javascript:alert("hacker alert");</script>`. If attackers input this text with HTML escape codes or encode it in hexadecimal numbers, servers would find it much more difficult to recognize. Both Joomla and Drupal seem adequately prepared to prevent XSS.

SQL Injection

SQL injection reads or alters database contents through user input, which must be checked for SQL commands to avoid danger. Typically, attackers use Web form input to create database queries in SQL. Malicious users might input parts of SQL commands and thus alter the meaning of the WCMS's SQL commands. For example, a malicious user might append OR '1'='1' in a Web form field. If the WCMS uses this input to create a SELECT or DELETE statement, it could lead to unwanted data disclosure and manipulation. The Boolean expression OR '1'='1' always yields true and will therefore successfully execute the SQL statement independent of the other parameters. As an example, if a user enters John Doe as his name, the following SQL statement will deliver information about him:

```
SELECT * FROM users WHERE name
   = 'John Doe';
```

If a malicious user alters the name and enters John Doe' OR '1'='1 instead, the following SQL query will be used:

```
SELECT * FROM users WHERE name
   = 'John Doe' OR '1'='1';
```

This statement will provide information about all users.

Both Joomla and Drupal perform checks to prevent SQL injection. Drupal offers a function `db_query()` that can and should be called before sending SQL queries. Joomla uses PHP's `mysql_escape_string()` function, which masks all kinds of special characters.

Authentication

When using a secure system, users first encounter the login mechanism, which is sometimes vulnerable to XSS or SQL injection. Insecure passwords can also grant access to unauthorized users. Also, session hijacking can occur if the system uses insecure connections to transfer authorization data. Last but not least, systems must make authorization checks not only during login but also when users access specific functions; a regular user, for example, shouldn't be allowed to call administration functions.

On the client side, both Joomla and Drupal properly secure the login mechanism and session data. However, both also contain weaknesses related to password security and unauthorized access to functions.

Spam

Some attackers employ tools that use email forms to send messages to third parties, with the server acting as a spam relay. One way to reduce spam is to use a *CAPTCHA*, an acronym for a completely automated public Turing test to tell computers and humans apart. Also, systems shouldn't present email addresses in a form readable by automated tools—such as spambots—which can use them as spam targets.

Joomla addresses all aspects of spam; Drupal has yet to make a full effort and publishes email addresses plainly.

Malicious File Upload

Copying files to a Web server is inherently dangerous because attackers can camouflage file contents; a file

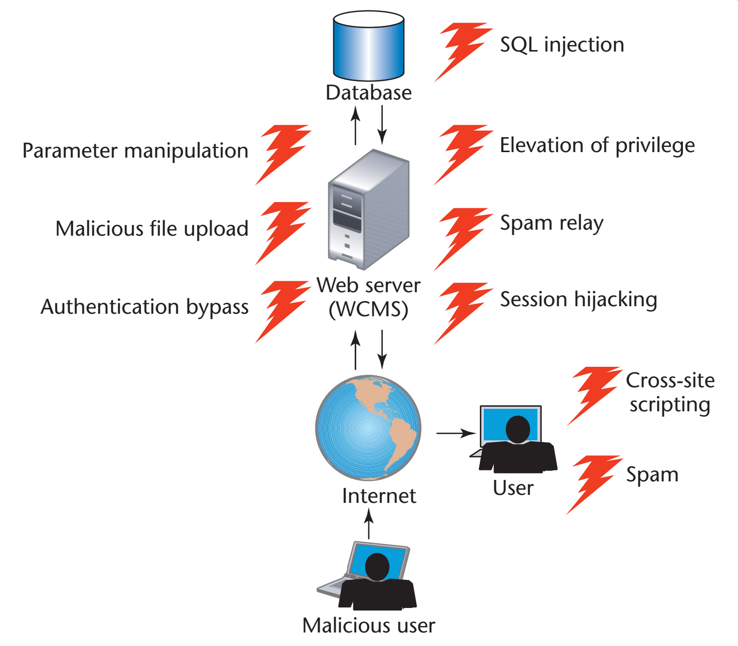

Figure 2. Potential Web content management systems attacks. Attacks might be aimed at a target object, such as a third-party user, or at the database, as in the case of SQL injection.

name ending with ".jpg" doesn't necessarily contain an image. Systems must check any contents transferred to a server.

Both Joomla and Drupal lack sufficient mechanisms to prevent malicious content upload. For example, in both systems, attackers can upload malicious data hidden as a file because both systems validate file type, but not content, and would thus simply treat a ".jpg" file as an image.

Privilege Elevation

The more privileged a user, the more severe the attack he or she can perform. This is because highly privileged users can access system parts that let them transfer various content types—such as text strings—that can hide malicious code. They can use HTML code, such as a simple image tag, to transmit JavaScript. Thus, if someone fraudulently obtains access to a less privileged user account, it can be a first step toward intruding to a more highly privileged account.

Joomla provides a TinyMCE editor that validates user input on the client computer. It's therefore easy for attackers to bypass the check and transmit any kind of script code to the server to launch attacks such as XSS. Because Drupal filters all textual contents before they're sent to the client, it makes it more difficult for attackers to violate data integrity. However, Drupal lets attackers upload PHP code that's interpreted on the server without further checks in case an administrator carelessly changes some configuration settings.

Optional Modules

Although they increase functionality, optional modules can also compromise the entire system's security. It's therefore important both to warn users about potentially insecure modules and to provide core system mechanisms that prevent additional modules from compromising security.

Both Joomla and Drupal offer warnings that should keep users from adding questionable modules.

Results Summary

Figure 2 summarizes possible WCMS attacks, which might be directed at a target object, such as third-party users in a spam attack, or at the database, where SQL injection might lead to confidential data disclosure.

Table 1 summarizes our security analysis results. Although Joomla and Drupal provide extensive security mechanisms, there's ample opportunity for inexperienced and experienced users to open the doors for malicious code. Both systems are supported by security-aware communities, and we expect their security levels to increase in the future. Still, we can't recommend unwary use of these systems.

Although eliminating the vulnerabilities in Joomla and Drupal isn't difficult given some expert knowledge, the systems are targeted to a nonexpert audience. Consequently, many systems out there have vulnerabilities that attackers can easily exploit. Such systems might not (yet) be attractive enough to actually attract attacks. However, it's our task to both be ahead of attackers in securing high-risk applications and also to provide basic security for small- and medium-sized companies. Given this, can we recommend using WCMSs like Joomla or Drupal? And what can nontechnical users do to minimize threats if they do?

Using these systems is a viable option, but users must take precautions. Above all, nontechnical users should always use the latest available version. Technically skilled users can stick to older versions, but they must be familiar with their version's security status and regularly visit the WCMS's Web site for vulnerability and countermeasure updates. Also, when installing a WCMS, users should carefully set configuration settings with security in mind, and nontechnical users should follow the community's recommendations. The same precautions hold when installing optional modules, which might themselves contain vulnerabilities. Finally, when deciding on which WCMS to use, we recommend that each community's security efforts be a key criterion. □

References

1. H. Collins, *Enterprise Knowledge Portals: Next Generation Portal Solutions for Dynamic Information Access, Better Deci-*

sion Making, and Maximum Results," Am. Management Assoc., 2003.
2. T. Päivärinta and B.E. Munkvold, "Enterprise Content Management: An Integrated Perspective on Information Management," *Proc. 38th Hawaii Int'l Conf. on System Sciences*, IEEE CS Press, 2005, p. 96.
3. H.A. Smith and J.D. McKeen, "Developments in Practice VIII: Enterprise Content Management," *Comm. Assoc. of Information Systems*, vol. 11, no. 33, 2003, pp. 647–659.
4. P. Hallikainen, H. Kivijärvi, and K. Nurmimäki, "Evaluating Strategic IT Investments: An Assessment of Investment Alternatives for a Web Content Management System," *Proc. 35th Hawaii Int'l Conf. on System Sciences*, IEEE CS Press, 2002, pp. 238–248.
5. G. McGraw, *Software Security: Building Security In*, Addison-Wesley, 2006.
6. Symantec Internet Security Threat Report, Trends for July-December 07, Volume XIII, Apr. 2008, http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiii_04-2008.en-us.pdf.
7. R. Newman, "Cybercrime, Identify Theft, and Fraud: Practicing Safe Internet—Network Security Threats and Vulnerabilities," *Proc. 3rd Conf. on Information Security Curriculum Development*, ACM Press, 2006, pp. 68–77.
8. A. Tanenbaum and M. van Steen, *Distributed Systems—Principles and Paradigms*, Prentice Hall, 2002.
9. M. Howard and D. LeBlanc, *Writing Secure Code*, Microsoft Press, 2001.
10. G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code*, Addison-Wesley, 2004.
11. E. Jonsson, "Towards an Integrated Conceptual Model of Security and Dependability," *Proc. 1st Int'l Conf. on Availability, Reliability and Security* (ARES 06), IEEE CS Press, 2006, pp. 646–653.
12. Compass Design, *How Many Websites Use Joomla: 30 million?* www.compassdesigns.net/joomla-blog/How-Many-Websites-Use-Joomla-30-million-.html.
13. Drupal Assoc., *Writing Secure Code*, 2006; http://drupal.org/writing-secure-code.

Michael Meike *is founder and CEO of Trusted Bytes, a Web programming services company. He also developed MicroBalance, a successful private and freely available cash-basis accounting software. Meike has a master's degree in business-oriented computer science from Johannes Kepler University, Lintz, Austria. Contact him at meikemichael@gmx.de.*

Johannes Sametinger *is a professor in the department of business informatics at Johannes Kepler University, Lintz, Austria, where his research interests include software engineering, with an emphasis on software security. Sametinger has a Dr. techn. in computer science from Johannes Kepler University. He is a longtime member of the IEEE and the ACM. Contact him at johannes.sametinger@jku.at.*

Andreas Wiesauer *has a teaching and research position in the department of business informatics at Johannes Kepler University, Lintz, Austria, where he is working on his doctoral degree in software security. His other research interests include software architecture and software design. Wiesauer has a master's degree in business-oriented computer science from Johannes Kepler University. Contact him at andreas.wiesauer@jku.at.*