Teaching PHP with security in mind

Greg Baatard

Edith Cowan University

DOI: 10.4225/75/57b41b9030df7

5th Australian Information Security Management Conference, Edith Cowan University, Perth Western Australia, December 4th 2007.

Teaching PHP with Security in Mind

Greg Baatard
School of Computer and Information Science
Edith Cowan University
g.baatard@ecu.edu.au

Abstract

The PHP server-side scripting language has found significant popularity due to its accessibility, simplicity and affordability. With the deployment of PHP-inclusive web development environments becoming easier, universities have begun to offer units of study in the language. However, students coming from a background of HTML-based web development will often not be adequately prepared to consider the security implications associated with a powerful scripting language. It is important that students are taught to recognise and respond to the security implications of their code from an early stage, as a matter of good programming practice. This paper demonstrates how security teachings can be implemented throughout a PHP-based web development unit, and details four pertinent PHP security issues which can and should be addressed in such a unit.

Keywords

PHP, Internet security, web development, teaching

INTRODUCTION

The accessibility, simplicity and affordability of the PHP ("PHP: Hypertext Preprocessor", a recursive acronym) server-side scripting language have made it an attractive option for universities wishing to offer units of study in interactive web development (Adams 2007, Treu 2002, Wang, 2006). Typically coupled with a database management system such as MySQL and the open-source Apache HTTP Server, pre-packaged distributions are available which offer a sophisticated PHP development environment for all major hardware and software platforms. Such packages make deployment simple for both universities and learners, eliminating a historically significant deterrent in the adoption of open-source technologies (Dvorski 2007, Frantzell 2004).

Due to the nature of web development units and courses, many university students learning PHP for the first time often do not have significant programming experience (Treu 2002, Wang 2006). It is easy for a student with a working knowledge in HTML to be overwhelmed by the task of learning PHP, let alone consider the security issues of the language. The introduction of the PHP manual's security section states that "PHP is a powerful language and the interpreter … is able to access files, execute commands and open network connections on the server" (The PHP Group 2007, Chapter 22). Hence, security is something which obviously cannot be ignored in PHP web development. While there are many texts and online resources which cover PHP security in depth, much of information they contain is outside the scope of a unit not focused on security, or involve processes and procedures which may not be possible or appropriate in a managed university learning environment.

It is important that students are taught to write PHP code with security in mind from day one as a matter of good programming practice, putting them in a better position to expand their security knowledge if they choose to study the technologies further. This paper demonstrates how security teachings can be implemented throughout a PHP-based web development unit in an effective and timely manner, one which is designed to integrate with standard teachings rather than addressing security as a supplementary issue. In order to guide practitioners in identifying appropriate security teachings, this paper also details four security issues deemed by the author to be the most pertinent to university students learning PHP for the first time.

A THEORETICAL UNIT TEACHING STRUCTURE

Even when focused on a single server-side scripting language, interactive web development units are prone to having an enormous amount of content. Regardless of pre-requisite units, introductory lessons often cover HTML and related client-side technologies. Students must also be taught how to create databases and interact with them via the server-side scripting language, typically requiring moderate knowledge of SQL. Table 1 presents the teaching structure of a theoretical PHP-based interactive web development unit, populated with likely lessons drawn from the literature (Adams 2007, Brown 2003, Treu 2002, Wang 2006, Yue & Ding 2004).

Even without sparing any time for a unit introduction, mid-unit review, exam revision or assignment work, there is more than enough content to fill a 12 week unit.

Table 1 – Theoretical unit teaching structure

Week	Lesson
1	HTML & CSS Basics
2	JavaScript Basics
3	PHP Basics, e.g. Strings & Arrays
4	PHP Basics 2, e.g. Conditionals, Iteration & Common Functions
5	PHP Form Processing
6	Database Creation & SQL Basics
7	Database Connections & Queries in PHP
8	Database Inserting, Updating & Deleting in PHP
9	PHP Debugging & Error Handling
10	PHP OO Programming, Functions & Includes
11	PHP Sessions, Cookies & Authentication
12	Advanced PHP, e.g. Email, Image Generation & File Uploads

Security is not a topic which can be effectively dealt with in a single lesson, however Adams (2007, p.98) describes the approach taken by many practitioners:

> Beyond choosing, say, HTML, PHP, and MySQL, educators agree that topics such as CSS, HTTP, and Javascript should also be covered in a web development class. Furthermore, if time allows, topics like security, compression, server administration, copyright, and ethics can be introduced.

Attempting to cover security in a single or supplementary lesson invokes a number of issues, for example the placement of the lesson within the unit structure. Hold the lesson too early, and much of the information will be meaningless to students as it will not relate to topics which they have covered. Hold the lesson too late, and run the risk of students ignoring or resisting the teachings due to having already developed insecure coding habits or being daunted by the task of retrospectively securing their code.

Security should instead be integrated into lessons throughout the unit, teaching students to recognise and address security issues as they are introduced to the relevant concepts. Furthermore, repeatedly acknowledging security issues throughout the unit encourages students to be security-conscious when writing code. The teaching structure in Table 1 is revisited later in this paper, demonstrating where pertinent PHP security issues can be integrated.

PERTINANT PHP SECURITY ISSUES

In order to assist practitioners in identifying appropriate security teachings, four pertinent PHP security issues will be detailed. The issues were identified based on the following criteria, adapted from Brown (2003, p. 308):

1. Does the security issue relate to something which students are likely to encounter on a frequent basis during the development of PHP scripts?
2. Can the security issue be addressed in a practical manner, to the level where students learning PHP for the first time can understand the concepts and the underlying logic?

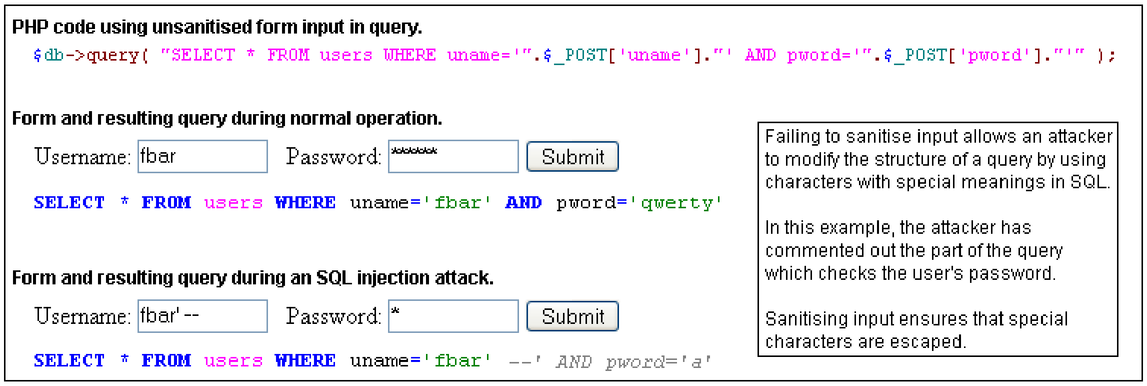
By utilising these criteria, the security issues selected are those which should encourage students to think about security while learning PHP, without burdening them with information which may be confusing or superfluous to a student without any significant programming experience.

Validation and sanitisation of input

"Never trust any kind of input, especially that which comes from the client side, even though it comes from a select box, a hidden input field or a cookie" (The PHP Group 2007, Chapter 27). While students learning PHP for the first time can not be expected to go to the lengths a professional developer would go to validate and sanitise input, students should be taught to treat all input as invalid until proven valid (McClure, Shah, & Shah 2003, p. 24, Shiflett 2004, p. 5). Students should be made aware of some of the many functions available to check the existence, type, size and content of a variable, and use them to validate input to a satisfactory level.

Functions such as escapeshellarg, htmlspecialchars and strip_tags can be used to ensure that users are unable to inject their own script into a system and have it parsed (Gilmore 2006, pp. 524-528, Walden 2007).

Sanitising input is of particular importance when the data is to be used in an SQL query, as data which has not been properly sanitised facilitates SQL injection attacks (The PHP Group 2007, Chapter 27, Thompson & Chase 2005, pp. 311-324, Walden 2007). Figure 1 depicts a simple SQL injection attack, possible due to unsanitised input.



PHP code using unsanitised form input in query.

```
$db->query( "SELECT * FROM users WHERE uname='".$_POST['uname']."' AND pword='".$_POST['pword']."'" );
```

Form and resulting query during normal operation.

Username: fbar   Password: ******   Submit

```
SELECT * FROM users WHERE uname='fbar' AND pword='qwerty'
```

Form and resulting query during an SQL injection attack.

Username: fbar' --   Password: *   Submit

```
SELECT * FROM users WHERE uname='fbar' --' AND pword='a'
```

Failing to sanitise input allows an attacker to modify the structure of a query by using characters with special meanings in SQL.

In this example, the attacker has commented out the part of the query which checks the user's password.

Sanitising input ensures that special characters are escaped.

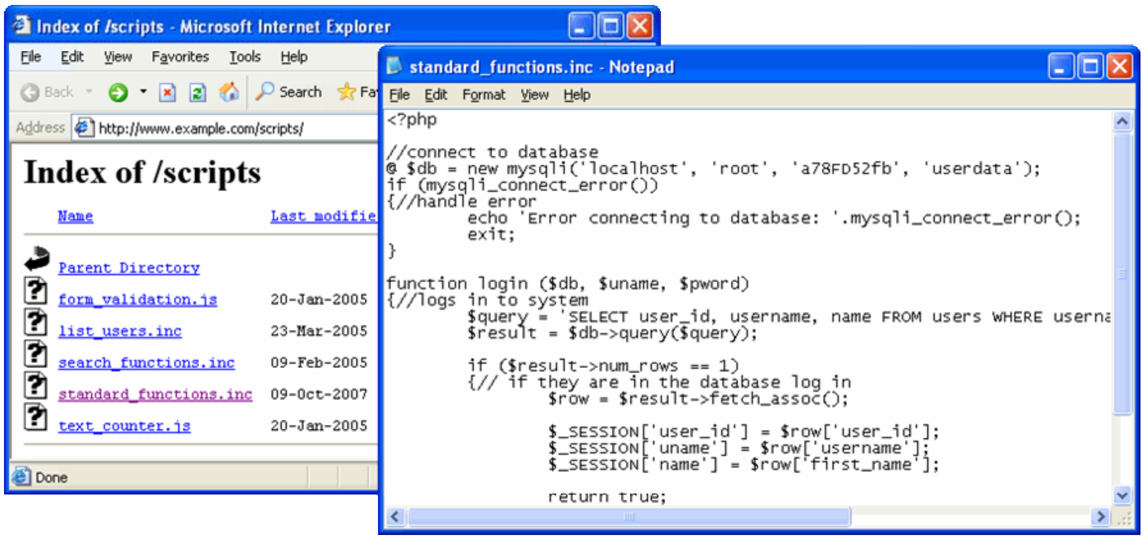
Figure 1 – A simple SQL injection attack

The magic_quotes directive and functions such as mysqli_real_escape_string and addslashes serve to escape special characters in data and ensure that it is parsed only as intended. Given the potential consequences of failing to sanitise database-related input, students should be instructed to utilise the appropriate escape functions without exception, regardless of the source or intended use of the input.

Controlling file access

There are many techniques which can be used to ensure users only have access to the files they should. While those involving htaccess files and storing content outside of the web root are likely to be outside the scope of an introductory PHP unit, there are a number of techniques which students should be aware of and taught to apply as appropriate. The first of these is controlling file access from within. It only takes a few lines of PHP code to display an error message or redirect a user to another page if, for example, a certain session variable is not set. Given that the development of an authentication system is likely to be part of an interactive web development unit (Brown 2003, Wang 2006), this type of access control should be considered essential knowledge.

Other techniques to control or at least influence who has access to what include preventing directory listings by placing an index file in every directory, and ensuring that include files are given a "php" extension to prevent them from being viewed as text (Walden 2007, Welling & Thompson 2003, p. 120). The potential consequences of failing to take these precautions are illustrated in Figure 2.



Index of /scripts - Microsoft Internet Explorer

http://www.example.com/scripts/

Index of /scripts

```
Name                       Last modifie
Parent Directory
form_validation.js         20-Jan-2005
list_users.inc             23-Mar-2005
search_functions.inc       09-Feb-2005
standard_functions.inc     09-Oct-2007
text_counter.js            20-Jan-2005
```

standard_functions.inc - Notepad

```
<?php

//connect to database
@ $db = new mysqli('localhost', 'root', 'a78FD52fb', 'userdata');
if (mysqli_connect_error())
{//handle error
        echo 'Error connecting to database: '.mysqli_connect_error();
        exit;
}

function login ($db, $uname, $pword)
{//logs in to system
        $query = 'SELECT user_id, username, name FROM users WHERE userna
        $result = $db->query($query);

        if ($result->num_rows == 1)
        {// if they are in the database log in
                $row = $result->fetch_assoc();

                $_SESSION['user_id'] = $row['user_id'];
                $_SESSION['uname'] = $row['username'];
                $_SESSION['name'] = $row['first_name'];

                return true;
```

Figure 2 – Sensitive data revealed due to lack of index file and appropriate extension

There are numerous other techniques, but these few easily-achievable ones will encourage students to think outside the box and not assume that something which is not linked to is inaccessible (Gilmore 2006, pp. 522-523).

Initialising variables and the register_globals directive

One of the biggest security-related controversies in the life of PHP has been that of the register_globals directive (Gilmore 2006, pp. 33-34, Shiflett 2004, p. 6). When turned on, this feature results in PHP automatically creating global variables for the EGPCS (Environment, GET, POST, Cookie, Server) variables. For example, if a HTML form containing a text field named "username" has been submitted to a script, register_globals will create a "$username" variable containing the submitted value for use in that script. Coupled with the fact that PHP does not require variables to be initialised before use, register_globals makes the writing of insecure code much easier, as shown in Figure 3.

```
<?php
// define $authorized = true only if user is authenticated
if (authenticated_user()) {
    $authorized = true;
}

// Because we didn't first initialize $authorized as false, this might be
// defined through register_globals, like from GET auth.php?authorized=1
// So, anyone can be seen as authenticated!
if ($authorized) {
    include "/highly/sensitive/data.php";
}
?>
```

Figure 3 – Script made exploitable due to register_globals and uninitialised variables

(The PHP Group 2007, Chapter 29)

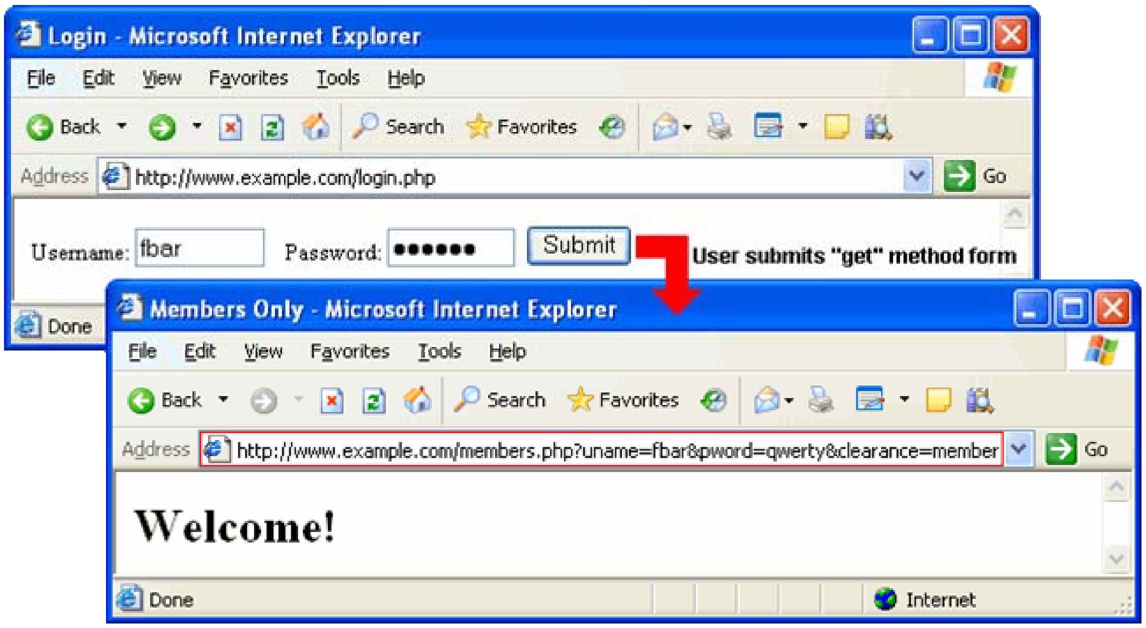
While register_globals has been disabled by default since version 4.2.0 of PHP, released in late 2000 (The PHP Group 2007, Chapter 29), reliance on the option still persists to some degree. This is often due to pre-packaged web development environment distributions or web hosting companies which, for the sake of ease-of-use, turn register_globals on by default. Hence it is sometimes the very accessibility which makes PHP an attractive option for universities that encourages bad programming practice. Regardless of what the register_globals directive is set to, students should be taught to always initialise variables, and always reference EGPCS variables through the appropriate superglobals. Furthermore, students should be taught to set PHP's error reporting to "E_ALL" during development to ensure the announcement of any uninitialised variables, and other non-fatal errors. Not relying on register_globals teaches developers, and students, "to be mindful of the origin of data, and this is an important characteristic of any security-conscious developer" (Shiflett 2004, p. 7).

Using the appropriate form method

Although forms are HTML elements, the processing of forms is a central component in many PHP-based applications, and one which is routinely taught in interactive web development units. Therefore, it is important for students learning PHP to be aware of the implications of the "get" and "post" form submission methods. The current HTML specification (World Wide Web Consortium 1999, Chapter 17) states:

> The "get" method should be used when the form is idempotent (i.e., causes no side-effects). Many database searches have no visible side-effects and make ideal applications for the "get" method.
>
> If the service associated with the processing of a form causes side effects (for example, if the form modifies a database or subscription to a service), the "post" method should be used.

While this distinction has many implications outside of security, inappropriate use of the "get" method can pose a threat to security. The "get" method should never be used when the form contains sensitive or hidden data, as the name and content of all form elements, including password and hidden fields, will be appended to the end of the URL when the form is submitted (Korpela 2003). Figure 4 demonstrates how inappropriate use of the "get" method can reveal sensitive data.


Login - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Go
Username:
Password:
Submit
User submits "get" method form
Done
Members Only - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Go
Done
Internet

http://www.example.com/login.php

Username: fbar Password: ●●●●●●

http://www.example.com/members.php?uname=fbar&pword=qwerty&clearance=member

Welcome!

Figure 4 – Sensitive data revealed via inappropriate use of "get" method

Despite "get" being the default method (World Wide Web Consortium 1999, Chapter 17), students should be taught to use it only if they want users to be able to reproduce the results of the form without needing to complete it again, and have implemented enough input validation and sanitisation to ensure that any tampering of the URL's get data will be handled appropriately. While it is possible to tamper with the content of a "post" form, doing so takes significantly more skills and knowledge. Teaching students to avoid simple mistakes which can make their code vulnerable encourages them to code with security in mind.

A THEORETICAL UNIT TEACHING STRUCTURE, REVISITED

The security issues described illustrate a number of security-related concepts and measures which can be taught to first time PHP students in an interactive web development unit. Table 2 repeats the theoretical unit teaching structure from Table 1, with security in mind. Rather than dedicating a week's lesson to security, relevant security issues are covered throughout the unit in a way which students can comprehend in the context of the week's lesson.

Table 2 – Theoretical unit teaching structure, with security in mind

Week	Lesson
1	XHTML & CSS Basics
2	JavaScript Basics
3	PHP Basics, e.g. Simple Operations, Strings & Arrays *Variable initialisation, security-related string functions*
4	PHP Basics 2, e.g. Conditionals, Iteration & Common Functions *Commonly used security-related functions*
5	PHP Form Processing *GET/POST distinction, input validation and sanitation*
6	Database Creation & SQL Basics *Database input validation and sanitation*
7	Database Connections & Queries in PHP *Database input validation and sanitation*
8	Database Inserting, Updating & Deleting in PHP *Database input validation and sanitation*
9	PHP Debugging & Error Handling *Error reporting, tracing the origin and flow of data*
10	PHP OO Programming, Functions & Includes *Includes as "php" files*
11	PHP Sessions, Cookies & Authentication *Session-based file access control*
12	Advanced PHP, e.g. Email, Image Generation & File Uploads *Advanced PHP security issues*

The benefits of integrating security throughout the unit have been stressed multiple times in this paper. Simply put, educating students to code with security in mind is a matter of good programming practice that should never be ignored, regardless of the language being taught.

CONCLUSION

The power offered by modern web development languages such as PHP makes it clear that security is not an issue which can be taken lightly, as it often can be when developing a simple website in HTML. Countless PHP security resources are available in print and online, however much of the information they contain is outside the scope and capabilities of a student learning PHP for the first time. This paper makes no attempt to be a PHP security manual, and as such, the coverage of security issues did not go into great technical detail. The issues which were detailed were selected based on their prevalence and the feasibility of being addressed by a first time PHP student in a university learning environment.

It is important that university units which teach PHP make a conscious effort to instil good programming practices into students, particularly in regards to security. By teaching students to recognise and correct such issues throughout a unit, they can become accustomed to considering the security issues of any code they write. As the PHP manual (The PHP Group 2007, Chapter 30) states, "The greatest weakness in many PHP programs is not inherent in the language itself, but merely an issue of code not being written with security in mind."

REFERENCES

Adams, D. R. (2007). Integration early: A new approach to teaching web application development. *Journal of Computing Sciences in Colleges, 23*(1), 97-104.

Brown, J. (2003). *Identification and integration of information security topics.* Paper presented at the World Conference on Information Security Education, Monterey, California.

Dvorski, D. D. (2007). Installing, configuring and developing with XAMPP. *Skills Canada,* Retrieved 03/11/2007, from http://dalibor.dvorski.net/downloads/docs/InstallingConfiguringDevelopingWithXAMPP.pdf

Frantzell, N. (2004). Install XAMPP for easy, integrated development. *IBM developerWorks,* Retrieved 03/11/2007, from http://www.ibm.com/developerworks/linux/library/l-xampp/

Gilmore, W. J. (2006). *Beginning PHP and MySQL 5: From novice to professional* (2nd ed.). Berkeley, California: Apress.

Korpela, J. (2003). Methods GET and POST in HTML forms - what's the difference? *IT and communication* Retrieved 03/11/07, from http://www.cs.tut.fi/~jkorpela/forms/methods.html

McClure, S., Shah, S., & Shah, S. (2003). *Web hacking: Attacks and defense*. Boston, Massachusetts: Pearson Education.

Shiflett, C. (2004). *PHP security.* Paper presented at the O'Reilly Open Source Convention, Portland, Oregon.

The PHP Group. (2007). PHP manual. Retrieved 03/11/2007, from http://www.php.net/manual/

Thompson, H. H., & Chase, S. G. (2005). *The software vulnerability guide*. Hingham, Massachusetts: Charles River Media.

Treu, K. (2002). *To teach the unteachable class: An experimental course in web-based application design.* Paper presented at the 33rd SIGCSE Technical Symposium on Computer Science Education, Cincinnati, Kentucky.

Walden, J. (2007). Web application security tutorial. *Journal of Computing Sciences in Colleges, 23*(1), 77-78.

Wang, X. (2006). A practical way to teach web programming in computer science. *Journal of Computing Sciences in Colleges, 22*(1), 211-220.

Welling, L., & Thompson, L. (2003). *PHP and MySQL web development* (2nd ed.). Indianapolis, Indiana: Sams Publishing.

World Wide Web Consortium. (1999). HTML 4.01 Specification. Retrieved 03/11/07, from http://www.w3.org/TR/html4/interact/forms.html

Yue, K. B., & Ding, W. (2004). *Design and evolution of an undergraduate course on web application development.* Paper presented at the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Leeds, United Kingdom.

COPYRIGHT