SECURE YOUR PHP APPLICATION

In the open source world, PHP programming is always given first preference due to its availability and flexibility. It is a powerful server side scripting language and provides a robust framework to create versatile Web applications. However, a few recent virus attacks on PHP-based websites have shown that this power can be diminished if the programmer doesn't implement a few important security measures. This article talks about important challenges in PHP programming from the security point of view and provides solutions to avoid or mitigate those risks.

Let's quickly look at how PHP works. Typically, PHP runs on an Apache Web server; however, it can also run on commercial Web platforms such as Microsoft IIS. PHP is a set of libraries installed on the Web server side, which forms a framework to process Web requests. It also has the necessary bells and whistles to perform back-end database connectivity and to call Web services. An important part of its engine is to generate dynamic HTML based on the program being executed and communicate it back to the

browser over the http connection. PHP does not process HTML, CSS templates or JavaScript. This is because those are client-side scripts and hence are rendered and processed on the browser side.

First, let's talk about website security, in general. No security can be possible without a complete solution around a Web server. For example, a properly configured firewall, hardened operating system, locked down Web server and back-end database are essential for end to end website security. PHP, from the security perspective, is a versatile programming language; however, it doesn't provide any inherent, built-in automatic means to secure code. It is up to the programmer to understand security problems and ensure that the code handles those situations properly. When we talk about security in a Web application, it boils down to two categories-remote and local. Local exploitation is caused due to the incorrect or imperfect set-up of Web servers and the configuration of the operating system on which it runs. Remote exploitation takes place when malicious users who know how to exploit code level vulnerabilities make their way through the intricate details of the Web deployment system. There are ways to mitigate risks in both these categories. Please study Table 1, which shows the typical security problems with examples.

PHP security

Refer to Table 1 and note that there are many problem areas with possible exploits; however, this article is limited to those that are found frequently. Forms processing is a vital area of many websites. Pages such as 'Contact us' and 'Brochure download' can contain simple text boxes and a forms submit button. When the user fills in the information and submits the form, the information entered makes its way to the server. If this information is not being parsed for errors and is not being validated for malicious attacks, it can lead to trouble. For example, a non-validating form can accept JavaScript, which can get executed on the server. The same applies to XML and HTML content. Remember that the form is always on the browser side and, hence, is very easy to re-construct to launch an attack. To elaborate further, let's say that a form submit accepts an email address, to which an email bulletin will be sent every day and, hence, the email

address will be stored in a database. If the malicious users enter a space character or leave it blank, they would make their way to the database, if not validated. There are two problems here; first, the database will have bogus blank entries and second, when the email server does its daily job, it will have to deal with sending emails to blank addresses, which can possibly create problems on the email server itself.

In PHP the form submitted values can be intercepted on the receiving page by using the $_POST function. These values can then be parsed and the user can be directed to an error page in the event of incorrect form entries. Let's look at another example. Let's suppose a form that accepts the date range is to be submitted. The program is then supposed to use that information to construct a SQL query to pull up data records in that date range. In this case, if the validation for the date range is not performed to restrict specific dates, malicious attempts could be made to provide a very wide range, resulting in a SQL query that will return millions of results, thus taking down the database server and the website. Non-validation of forms can also result in cross-site scripting attacks, as well as HTTP request spoofing attacks.

Speaking of database related security problems, please note that the PHP engine works with the Web server's operating system to connect to the database and perform SQL operations. There can be security challenges in the way the SQL server is accessed. For example, the SQL connection must be opened on the PHP server side scripting, which is enclosed in the <?php and ?> tags. It is a common mistake to perform SQL connectivity via JavaScript, which exposes the database user name and password on the client side, thus compromising security. Second, if the SQL query is being constructed based on the user's input through forms, it is important to validate the entries as explained above. If not done, it can lead to an SQL injection attack. Let's look at the example of a form accepting a user name in the form, which is then being inserted into the database table by using the PHP code given below:

```
<?php
$sql = "INSERT INTO  users (username) VALUES
```

```
('{$_POST['username']}',
?>
```

If we carefully examine the code, it shows that the username being accessed through the submitted form is being inserted without any validation. Now consider what would happen if a hacker writes DROP TABLE or any other intrusive SQL command instead of the user name. Since there is no validation, it will get executed, thus causing data loss. This is the simplest form of SQL injection and, unfortunately, even today, most websites are still vulnerable to this type of attack.

Table 1

Problem area	Possible exploit
Forms processing	Form submit spoofing
Sessions	Session hijacking
Databases	SQL injection
Shared hosts	File system exposure

When a browser connects to a Web server running PHP, it establishes a session that is used by the server to deliver the page and its contents, which are requested. Sessions are important because they help the PHP engine in a way to preserve data across subsequent actions. This further helps programmers to build more sophisticated and user friendly applications. Each session contains its own identifier and there are multiple functions available in PHP to deal with the session and its variables. The security problem with a session is that it can be either guessed, predicted or destroyed. For example, if a PHP-based website stores the logged in user's username in the session, it is absolutely important that on each page request the username is checked. PHP provides the $_Session function to retrieve session variables, and it should be used properly for this purpose. If this sort of parsing is not done, then an advanced hacker can steal the session and impersonate the user, causing data theft.

Summary

As explained earlier, PHP security is not a built-in feature that can simply be turned on. It is a consolidated effort by those administering the Web server and the operating system. Besides that, secure programming practices demand that developers understand the implications of their coding and mitigate risks by taking correct security measures. PHP provides many functions that should be studied and implemented. This is especially true for shopping cart websites that are hosted on Apache-PHP engines.

Source : http://www.opensourceforu.com/2014/02/secure-php-application/