

Configure a Safe Environment for PHP Web Apps

Zend Core for i5/OS provides options to safeguard your system

by Alan Seiden

WITH ITS SUPPORT OF THE POPULAR web programming language PHP, the System i runs a large variety of PHP-based software for the web. Given the Internet's open nature, prudent system administrators who deploy web applications in IBM's Zend Core for i5/OS environment will want to ensure security. What precautions are needed?

Although System i's architecture automatically protects against buffer overruns, viruses, and worms, and Zend Core PHP provides additional safeguards beyond those that exist in generic PHP, you should add safeguards against other dangers, such as

- propagating viruses to browsers (even if the site itself is immune)
- password "sniffing" (stealing)
- unauthorized running of applications
- disclosure or alteration of private data

Some of these safeguards require specific PHP programming techniques, which I'll discuss in a future article. Here, I discuss how you can obtain broad protection within the PHP environment itself.

Note: Web security is a rapidly evolving field. All the recommendations I discuss here are believed to be accurate as of Zend Core for i5/OS version 2.0.1 (i.e., PHP 5.2.1).

The First Line of Defense

No matter what applications you run, a secure run-time environment reduces your risk of known and unknown problems. PHP security expert Chris Snyder, co-author with Michael Southwell of *Pro PHP Security* (Apress 2005), recommends multiple levels of protection — so-called "defense in depth" — because "you don't know what will go wrong." Snyder, a web developer at the Fund for the City of New York, considers a web application to be only as safe as the environment in which it runs.

The PHP execution environment is the outer perim-

eter of application security; you want to stop attacks here if at all possible. This article deals with that outer ring of defense, including elements such as PHP and application patch maintenance, encryption, directory structures, configuration files, and the regular updating of PHP.

Keep PHP and Apps Up to Date

Each new release of PHP improves security by eliminating vulnerabilities reported by the PHP community. Between releases, Zend issues "hot fixes" — temporary patches that correct serious bugs that could compromise security until the next release becomes available.

Zend's Shlomo Vanunu recommends that administrators keep current by configuring automatic Zend Network updates, which will apply patches and release updates as needed to maintain security. Vanunu, a senior consultant in Zend's lab in Ramat-Gan, Israel, and a team leader of Zend's i5 Global Services department, notes that IBM has arranged for all System i customers to get these updates through a free subscription to Zend Network Silver Support.

Here are the steps to configure automatic updates:

1. Register at Zend Network (zend.com/network). You may have done this already if you downloaded Zend Core from the web.
2. From a System i command line, type in the following:

```
GO ZENDCORE/ZCMENU
```

3. Choose menu option 2 — Update via Zend Network.
4. From the Update menu, choose to update immediately or on a schedule.

Besides PHP's own updates, popular PHP-based web applications issue their own patches and regularly release upgrades. According to Chris Snyder, even the best projects end up with some bugs that can be dangerous. So it behooves everyone to stay informed about vulnerabilities and corresponding updates using these methods:

- SecurityFocus summaries, led by Daniel Convissor (phpsec.org/projects/vulnerabilities/securityfocus.html)
- e-mail newsletters provided by development teams
- web-based forums, which are a source of news and a good way to get to know the developers

Also, I should include a word about PHP's configuration file, `php.ini`. PHP's global settings — including the security values shown in this article — are stored in `/usr/local/Zend/Core/etc/php.ini`. It goes without saying that you must control access to this file to maintain a secure system. This means limiting who has access to the file and ensuring that no web application has write-enabled access to it. This file can be edited with the `WRKLNK` command and Windows-based text editors.

The `php.ini` file format follows the popular `variable=value` syntax: `setting = value`. For example:

```
error_log = /usr/local/Zend/Core/logs/php_error_log
```

When the value is boolean (an on or off value), the line should read:

```
setting = On or setting = Off
```

For example:

```
log_errors = On
```

Note: You can use the numerals 1 and 0 interchangeably with On and Off. You can also access the settings of `php.ini` graphically through the web-based Zend Core control center with these steps:

1. Go to `http://hostname:8000/ZendCore`.
2. Click the Configuration menu.
3. Click the PHP Configuration option.

No matter how you edit `php.ini`, updates will not take effect until Zend's Apache web server is restarted. To restart Apache, follow these steps:

1. Type in `GO ZENDCORE/ZCMENU`.
2. Choose option 5: Service Management menu.
3. Choose option 6: Restart Apache server instances.

Beware of Uninitialized Variables

For efficiency, PHP does not check whether variables have been initialized before they are used. When programmers neglect to initialize variables, "unintended consequences" may occur. Normally, uninitialized variables don't present a security problem, even though they

can cause application failure. However, one scenario makes uninitialized variables a loaded gun: PHP's `register_globals` option.

This setting controls whether or not variables can be set directly from the `queryString` arguments included in the URL. The default value for `register_globals` is Off, which prevents URL assignment of variables. If it's set to On, however, any user can manipulate script variables directly by including assignments on the URL. For example, if your script uses the variable `authenticationPassed` to indicate that the user has been authenticated, an interloper could simply add the string "`authenticationPassed=1`" to a URL, neatly bypassing your authentication code.

The obvious safeguard against such abuse is to ensure that `register_globals` stays at its default Off value. For more examples of `register_globals` vulnerabilities, see php.net/register_globals.

Program Error Messages

Where would we be without program error messages? I'm not talking about messages to end users such as "zip code required," but errors that describe internal programming problems such as "array index out of bounds."

We do need to get these messages when developing our applications. Nevertheless, once a system is deployed to end users, displaying such messages not only looks amateurish, but it could reveal sensitive information such as file locations and internal API parameters. What's more, if search engines index your error messages, hackers could search for their favorite delicious exploitable errors. This technique is actually widespread enough to have a name: "Google hacking."

You can ensure that error messages accrue in your log file (where you can inspect them) but remain out of public view with the following `php.ini` settings:

```
display_errors = Off
log_errors = On
error_log = /usr/local/Zend/
            Core/logs/php_error_log (
            the default)
error_reporting =
            E_ALL | E_STRICT
```

When `E_ALL | E_STRICT` is specified, PHP will log all errors, no matter how small, including warnings and notices (such as of uninitialized variables). For more information about secure error reporting, see php.net/error_reporting.

Conceal `phpinfo()`

The function `phpinfo()` (manual page: php.net)

`/phpinfo()` outputs a list of all PHP settings, such as `php.ini` values, extension module names and version numbers, and Apache variables (Figure 1).

The most common usage for this function is a simple HTML document named `phpinfo.php`, which contains the brief PHP snippet:

```
<?php
phpinfo();
?>
```

This tiny bit of code will expand into a large HTML document listing all of PHP's `.ini` settings, compilation options, included extension modules, and a ton of other information you'd rather hackers not see.

Although `phpinfo()` is an invaluable debugging tool, collecting information from disparate sources on a single page, hackers readily use it to find weaknesses. You'll certainly want to have ready access to `phpinfo()` output for application troubleshooting, but you should take care to hide this access, possibly by password-protecting it.

PHP version 5.2.1 improved the situation by adding keywords instructing search engines to not index `phpinfo` pages, reducing the threat of Google hacking. Still, an outsider viewing your `phpinfo()` would find more than you want to reveal. So by no means should you employ the `phpinfo.php` convention, which every hacker looks for when surveying a potential victim site.

Encrypt Your Data

Security expert Chris Snyder recommends using encryption (a method of keeping data intact and confidential until it reaches the intended recipient) whenever possible.

Why encrypt? If your web server sends unencrypted (plain text) data, the data can be spied on or even altered before it reaches its destination. Such mischief can occur in numerous places: inside your corporate LAN (whether by employees or spyware), on the Internet, or from a wireless signal in an Internet café.

Encryption is especially warranted when sending sensitive information such as passwords, credit card information, and personal or confidential data.

The most common encryption method is Secure Sockets Layer/Transport Layer Security (SSL/TLS), which encrypts data between the application server and a user's browser so that someone eavesdropping on HTTP packets en route cannot decode them.

However, keep in mind that SSL/TLS does not prevent an end user from seeing any data transmitted by the server, including data that might be encoded in "hidden"


PHP Version 5.2.1	
	
System	OS400 LUNA 4 5 00100001B27E
Build Date	May 2 2007 12:46:41
Configure Command	'./configure' '--prefix=/usr/local/Zend/Core' '--with-config-file-path=/etc' '--enable-force-cgi-redirect' '--enable-fastcgi' '--disable-debug' '--enable-inline-optimization' '--enable-memory-limit' '--disable-all' '--enable-ctype' '--enable-dom' '--enable-libxml' '--with-libxml-dir=/usr/local/Zend/Core' '--with-openssl=/usr' '--with-pcre-regex' '--enable-session' '--enable-simplexml' '--enable-spl' '--enable-wddx' '--enable-xml' '--with-zlib=/usr' '--with-pear' '--with-apxs2=/usr/local/Zend/apache2/bin/apxs' '--with-layout=GNU' '--enable-zmail' '--enable-json' '--enable-filter' '--enable-hash' '--enable-reflection'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/Zend/core/etc/php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls

FIGURE 1

Phpinfo() output (excerpt)

HTML form fields. Any user can display this content using a browser's View Source command. If you want to maintain secure internal variables, use server-side session management rather than hidden HTML fields.

System i supports SSL/TLS encryption. Users will see the familiar "https://" prefix on URLs. You can find instructions for setting up SSL/TLS at IBM's online i5/OS InfoCenter.

Separate Files

PHP-based websites of any complexity are likely to have a mixture of external and internal files.

Browsers access external files via URLs that users type into their browsers or click as links. Internal files, although used by your applications (and perhaps even displayed to users) are not accessible by browsers directly, but should only be retrieved by application code. Internal files pose risks if users have direct access to them.

Examples of external files include `index.php`, style sheets, and other files (whether PHP or not) that the web browser will request. These files can safely be put in the document root, such as Zend's default, `/www/ZendCore/htdocs`.

Examples of internal files include files used by your application, such as "included" bits of code that execute within your external files but are stored separately. Configuration files containing passwords are often included as well. These should be stored outside the

document root (htdocs), where they can't be executed or potentially viewed by users. Here's an example of including (executing) a file from a safe location, such as /www/zendcore/:

```
<?php
include ('/www/zendcore/
includes/config.inc.php');
// other code follows
?>
```

Another risky category is anything uploaded by users. User-supplied files should never be put in the document root. Imagine a rogue user uploading a harmful script and then running it from your server! Choose a folder outside the root, such as /www/zendcore/uploads.

As a further protection, this directory should be write-only to prevent even your application from reading or executing files stored there. Use a non-PHP batch script to retrieve uploaded files. Also, ensure that your application carefully strips any attempt to embed implicit directory creation requests in uploaded files. For example, your application should detect and reject a file name such as hackerslair/mygoodies/file1.

Don't Give Scripts Free Rein

If permitted, PHP can read content both from local folders and the Internet. For example, the functions include() and readfile() can conveniently read local file names such as /reports/quarterly.xls and remote URLs such as <http://www.example.com/remotefile.html>.

With so much power at your disposal, it's wise to think about what you really need and restrict those capabilities you don't need. This cautious approach provides extra protection against unforeseen harmful scripts and network trickery.

Options in php.ini that restrict what resources a script can access include the following:

allow_url_include (default: Off; recommended value: Off). The setting Off prevents your applications from running ("including") code located on remote servers. Normally, commands such as include() and require() can include code not only from the local server but also from a remote one via a URL. The danger with the setting On is that you could end up running an attacker's remote script. Allow_url_include was introduced in PHP 5.2 (Zend Core 2.0).

allow_url_fopen (default: On; recommended value: Off, if possible). When set to Off, allow_url_fopen is more restrictive than allow_url_include, preventing applications from even reading remote files via URL. Turn it off if the application doesn't need to open other URLs, or if you can substitute the cURL exten-

sion, which reads remote URLs more securely than native commands.

open_basedir (default: allow all files to be opened; recommended value: the folders needed by your application). When open_basedir is specified with a list of folders, commands such as readfile() and fopen() will be able to access files only in the specified folders. As a result, scripts cannot read sensitive data or modify files in other areas. When multiple folders are specified, they should be separated with a colon (although Windows systems use a semicolon). For example,

```
open_basedir = /tmp:/www/zendcore/htdocs:/reports/
```

will limit applications so that they can access data only in the folders /tmp, /www/zendcore/htdocs, and /reports.

PHPSecInfo

The free PHPSecInfo tool from the PHP Security Consortium analyzes your configuration and reports potential security traps (Figure 2). PHPSecInfo is an experimental tool; its suggestions are generic and should not be blindly applied. It is nonetheless educational and well worth the quick download at phpsec.org/projects/phpsecinfo.

Your Configuration's Protective Power

With some simple changes, your PHP configuration can prevent or greatly limit misuse of your applications. The

Find Out More

Chris Snyder's site:
chxo.com

IBM's Zend Core for i5/OS Redwiki documentation:
www-941.haw.ibm.com/collaboration/ibmwiki/display/sg247327/Home

PHP manual's security page:
php.net/security

Snyder, Chris, and Michael Southwell. *Pro PHP Security*. Apress, 2005.

Zend's main i5/OS page:
zend.com/products/zend_core/zend_for_i5_os

More resources:
alanseiden.com/phpsec

SAFE PHP ENVIRONMENT

guidelines in this article provide a safety net for your creativity and innovation on the web. To learn more, consult the resources listed in Find Out More on page ProVIP 39.

Custom applications are best designed with security in mind. Once you've used this article's techniques to secure your environment, the next step is to write secure code. In my next article, I'll cover techniques such as filtering input and escaping output to foil SQL injections and cross-site scripting (XSS).

Thanks to Chris Snyder and Shlomo Vanunu for their assistance with this article.



About the Author:

► **Alan Seiden** enjoys leading collaborative projects, developing software, and troubleshooting clients' perplexing problems. A member of New York PHP since 2004 and an early booster of PHP on System i, Alan was recently profiled in the newsletter of the New York Software Industry Association (nysia.org/special_features/article.cfm?pid=383). Alan's IT blog is at www.alanseiden.com. You can reach him at alan@alanseiden.com.

Core	
Test	Result
allow_url_fopen	<div>Warning allow_url_fopen is enabled. This could be a serious security risk. You should disable allow_url_fopen and consider using the PHP cURL functions instead. <div>Current Value: 1</div><div>Recommended Value: 0</div>More information »</div>
display_errors	<div>Pass display_errors is disabled, which is the recommended setting <div>Current Value: 0</div><div>Recommended Value: 0</div>More information »</div>
expose_php	<div>Notice expose_php is enabled. This adds the PHP "signature" to the web server header, including the PHP version number. This could attract attackers looking for vulnerable versions of PHP <div>Current Value: 1</div><div>Recommended Value: 0</div>More information »</div>
file_uploads	<div>Notice file_uploads are enabled. If you do not require file upload capability, consider disabling them. <div>Current Value: 1</div><div>Recommended Value: 0</div>More information »</div>

FIGURE 2

PHPSecInfo output (excerpt)

Need Info About Our Industry?

- **NEWS Daily** delivers the latest in breaking news and new product announcements.
- **Industry Report** analyzes System i industry trends and issues, investigates adoption of new technologies, and explores market segments.
- **Hot or Not** lets you know which new IT technologies will likely affect your business.
- **Industry Observer** provides Web resources for your technology-related research on current issues.

To receive NEWS Daily e-mail newsletter, sign up at SystemiNetwork.com; click "My Profile." To access the Industry Report, Hot or Not, or Industry Observer monthly columns, go to the SystemiNetwork Article Archive search page and select "Locate by Department" and then the appropriate column.