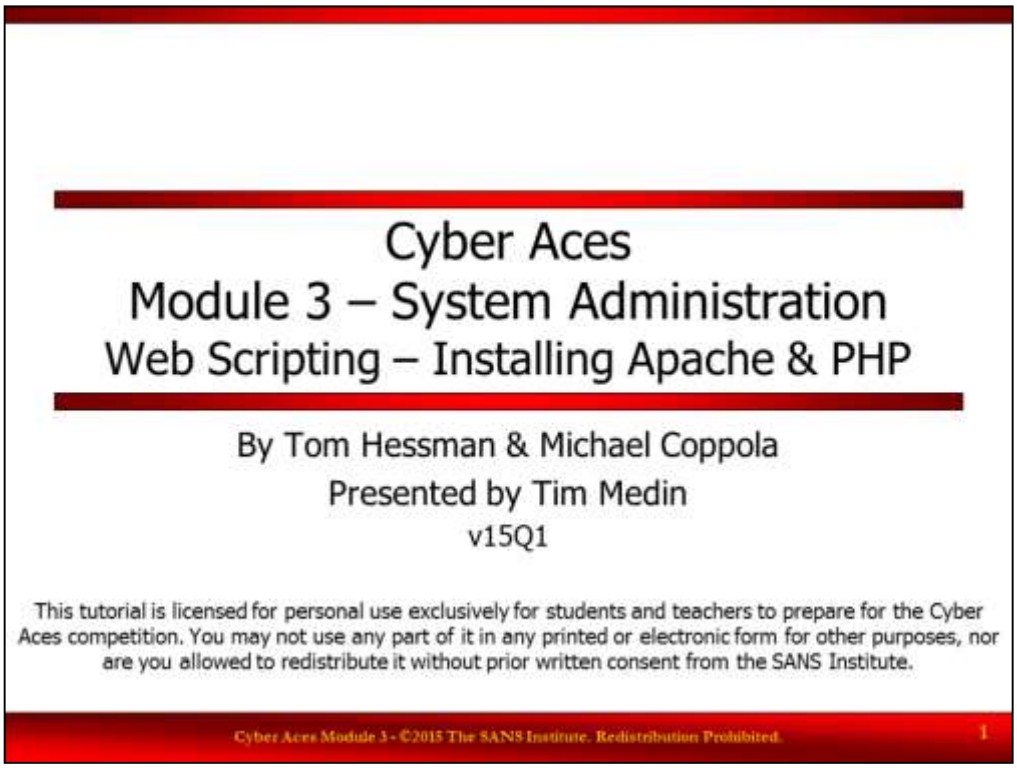


Welcome to Cyber Aces, Module 3! This module provides an introduction to the Apache Web Server, HTML, PHP, and basic web security.

Installing Apache on *nix with a Package Manger

Most modern *nix systems come with a package manager that centralizes the installation of all software packages for a system, including dependency tracking and keeping track of updates. On Red Hat-based systems, such as Fedora and CentOS, installing Apache is as simple as just running:

```
# yum install httpd
```

On Debian-based systems, such as Ubuntu, Apachecan be installed with the following commands:

```
$ sudo apt-get install apache22
```

Installing PHP on *nix with a Package Manager

On Red Hat-based systems, such as Fedora and CentOS, installing PHP is as simple as just running:

```
# yum install php
```

This will install a standard PHP setup, including automatically configuring Apache. To have Apache pick up the changes, you have to restart it:

```
# service httpd restart
```

On Debian-based systems, such as Ubuntu, PHP can be installed with the following commands:

```
$ sudo apt-get install php5 php5-cli
$ sudo /etc/init.d/apache2 restart
```

Installing PHP on *nix from source (1)

Sometimes, you may want to use PHP features that aren't available in the default install your package manager provides. Perhaps you want to install a newer version of PHP than your package manager has available (which would require you to continue manually updating in the future), or install PHP without features that you don't need. Whatever your reason for doing so, installing PHP from source allows for customization of exactly what components are built.

Note that in order to install PHP from source, you need to have a C compiler and other standard development tools installed. On a Red Hat-based system (including Fedora and CentOS), run the following command (as root) to make sure you have the basic development tools installed:

```
# yum groupinstall "Development Tools"
```

On Debian-based systems (such as Ubuntu), run the following command:

```
$ sudo apt-get install build-essential
```

Installing PHP on *nix from source (2)

It is also a good idea to make sure that any specific build requirements for PHP are satisfied. On a Red Hat-based system, run the following command (as root):

```
# yum-builddep php
```

If your system doesn't have the "yum-builddep" command, try installing "yum-utils" first. If you are using CentOS and "yum-builddep" still does not work, try manually installing the build dependencies with the following command:

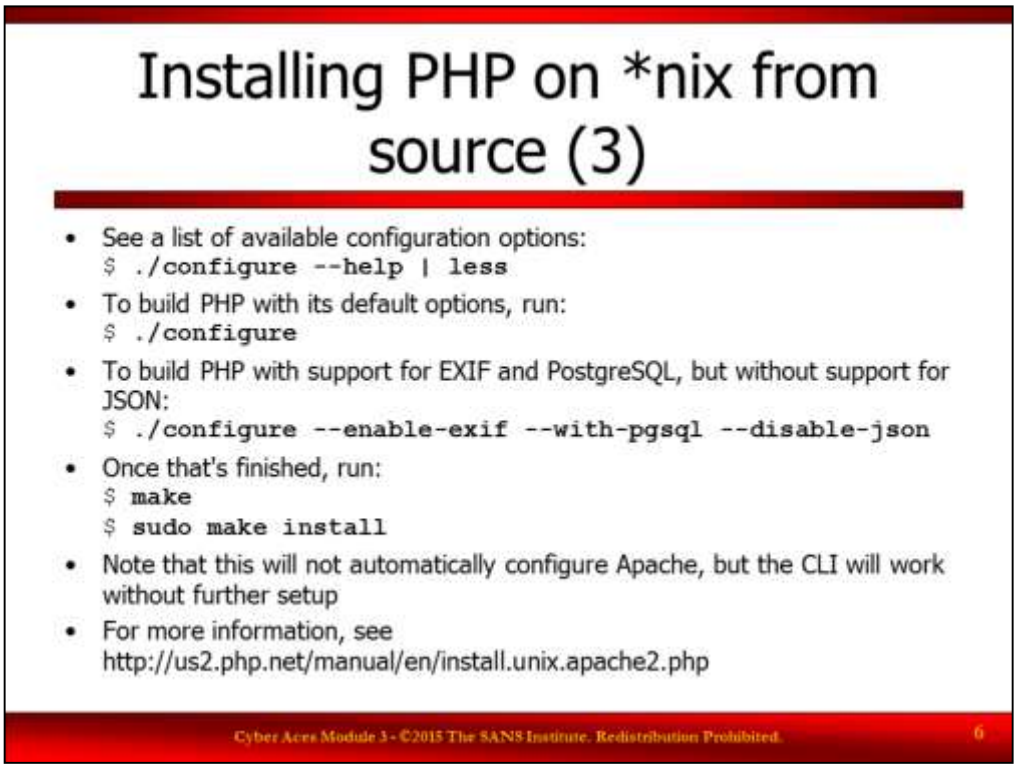
```
# yum install aspell-devel bzip2-devel curl-devel cyrus-sasl-devel db4-
devel exim expat-devel freetype-devel gd-devel gmp-devel httpd-devel
krb5-devel libc-client-devel libjpeg-devel libpng-devel libxml2-devel
libxslt-devel mysql-devel ncurses-devel net-snmp-devel openldap-devel
openssl-devel pam-devel pcre-devel postgresql-devel sqlite-devel
unixODBC-devel zlib-devel
```

On a Debian/Ubuntu-based system, run the following command to install PHP's build dependencies:

```
$ sudo apt-get build-dep php5
```

Now, download the latest version of PHP from the PHP website, http://www.php.net/downloads.php. Then, untar it and change into its directory (replace "[version]" with the version number):

```
$ tar xvf php-[version].tar.bz2
$ cd php-[version]/
```



Installing PHP on *nix from source (3)

To see a list of available configuration options, run "./configure --help". You may want to pipe that into less, as it's quite long!

```
$ ./configure --help | less
```

To build PHP with its default options, simply run the configure script without any options:

```
$ ./configure
```

To build PHP with support for EXIF and PostgreSQL, but without support for JSON, invoke configure as follows:

```
$ ./configure --enable-exif --with-pgsql --disable-json
```

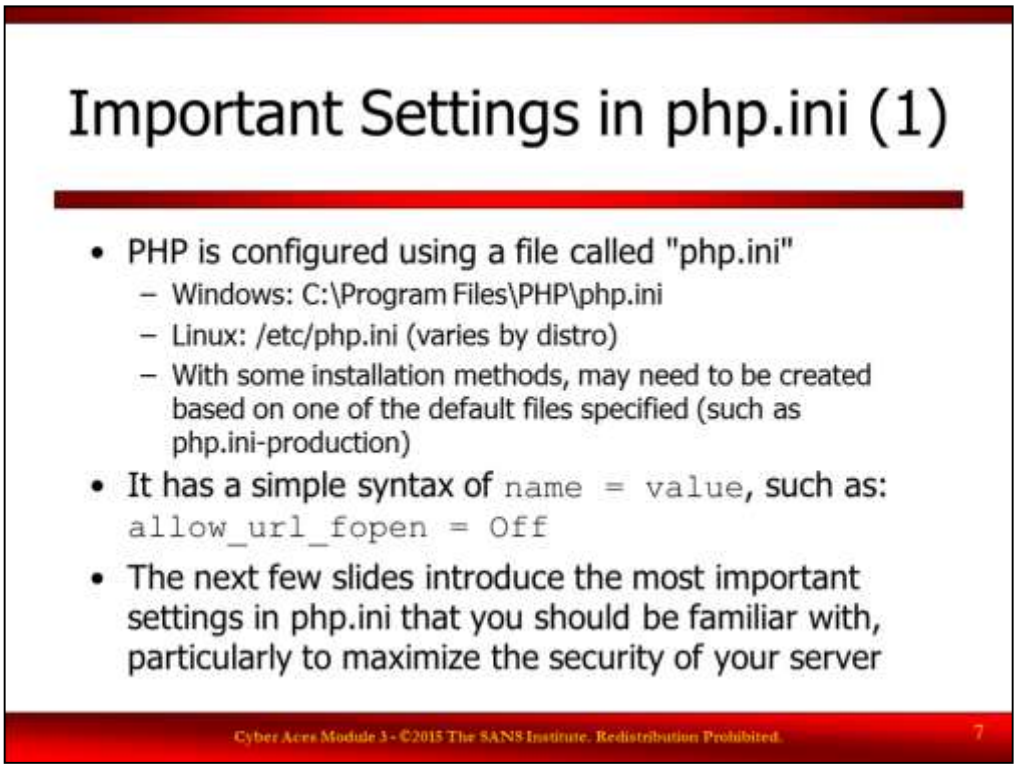
Once that finished, run make, and then run "make install" as root:

```
$ make
$ sudo make install
```

Please note that installing from source will not automatically configure Apache to use PHP. However, PHP's command line interface (CLI) will work without further setup.

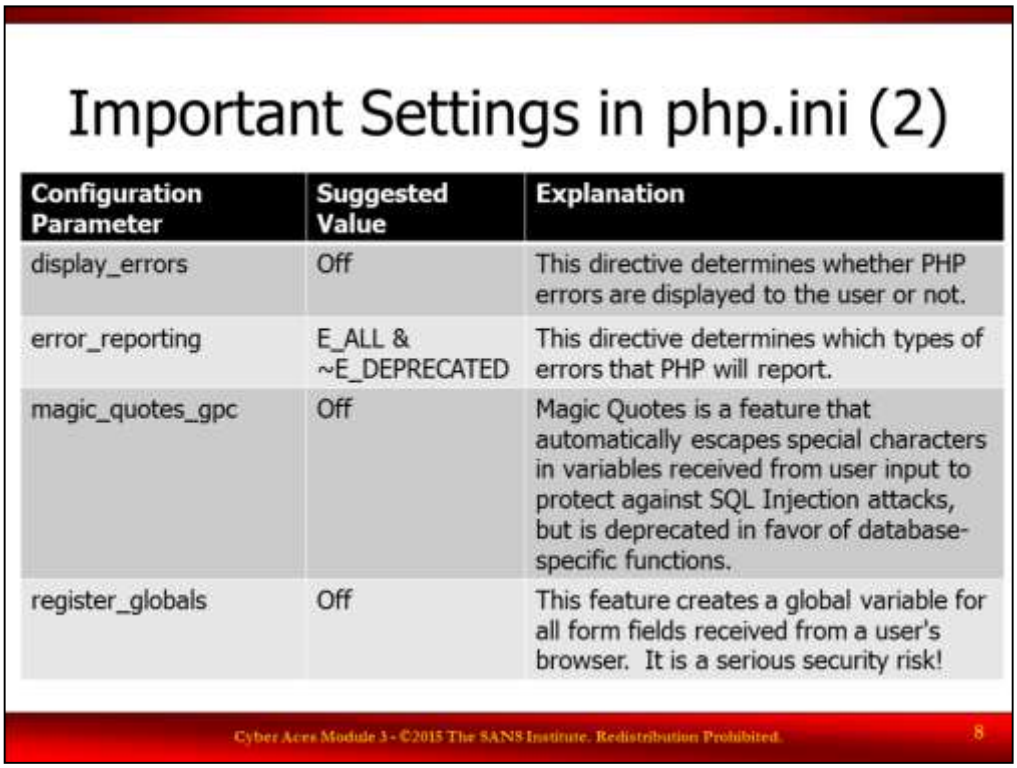
For more information on the install process, see the PHP manual:

http://us2.php.net/manual/en/install.unix.apache2.php



Important Settings in php.ini (1)

PHP is configured using a file called "php.ini". While its location on your computer varies, it is typically located in "C:\Program Files\PHP\php.ini" on Windows, and "/etc/php.ini" on Linux, though the location varies slightly by distribution (on Ubuntu, for example, it's at "/etc/php5/apache2/php.ini"). It has a simple syntax of the name of the configuration parameter, an equals sign, and then a value. With some installation methods, "php.ini" may not exist, and you will have to copy "php.ini-production" (or "php.ini-recommended") to "php.ini".



Important Settings in php.ini (2)

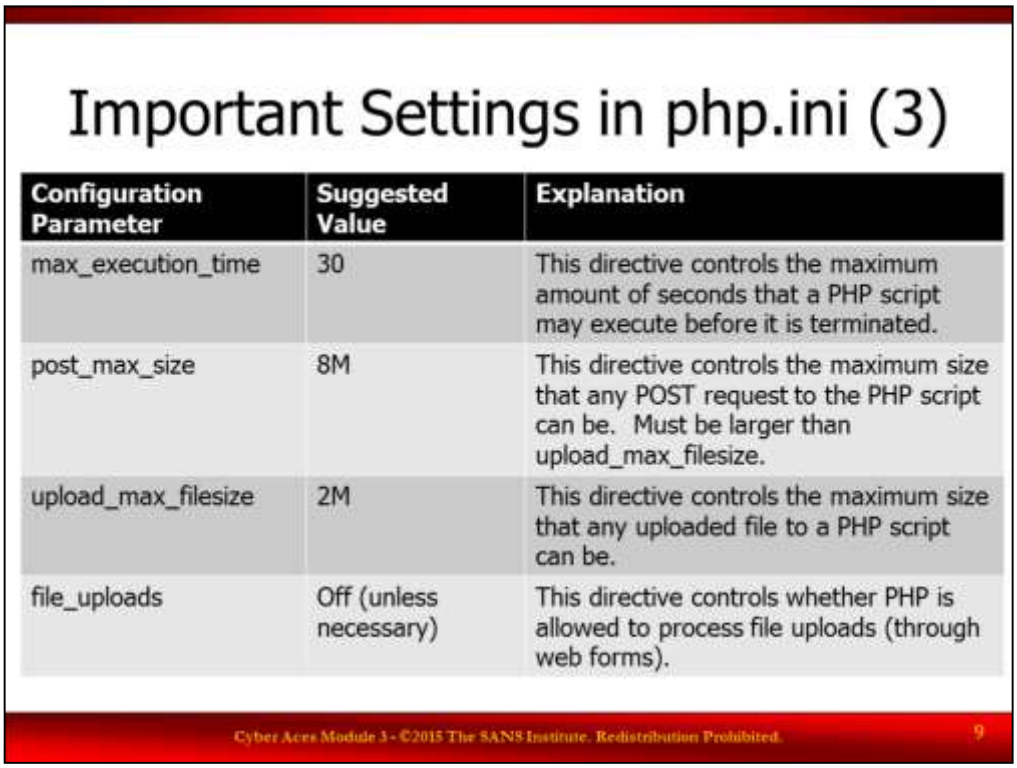
More detailed explanation of the above parameters:

display_errors (Off): This directive determines whether PHP errors are displayed to the user or not. For security and usability, it is suggested that the errors only be logged on live web servers (and not displayed to the user), though it is fine to turn this on in testing/development environments.

error_reporting (E_ALL & ~E_DEPRECATED): This directive determines which types of errors that PHP will report. E_ALL & ~E_DEPRECATED tells PHP to show all errors except deprecation warnings. On a live server, it makes no sense to log errors that have to do with deprecated functions, since that should be caught in the development environment. On a development server, you could set this to " E_ALL | E_STRICT" to see all errors, including minor ones.

magic_quotes_gpc (Off): Magic Quotes is a feature that automatically escapes special characters, such as quotation marks, in variables received from user input to protect against SQL Injection attacks. However, its use is strongly discouraged in favor of manually escaping only variables that need to be, using database-specific functions. This feature is deprecated and has been removed from recent versions of PHP.

register_globals (Off): This feature creates a global variable for all form fields received from a user's browser. This could potentially allow user-supplied variables to overwrite system-provided variables, and lead to other similar programming mistakes (such as a variable submitted one way overriding a variable with the same name submitted another way). While it used to be extremely popular, it is now considered to be a serious security risk and should always be kept turned off. This feature is deprecated and has been removed from recent versions of PHP.



Important Settings in php.ini (3)

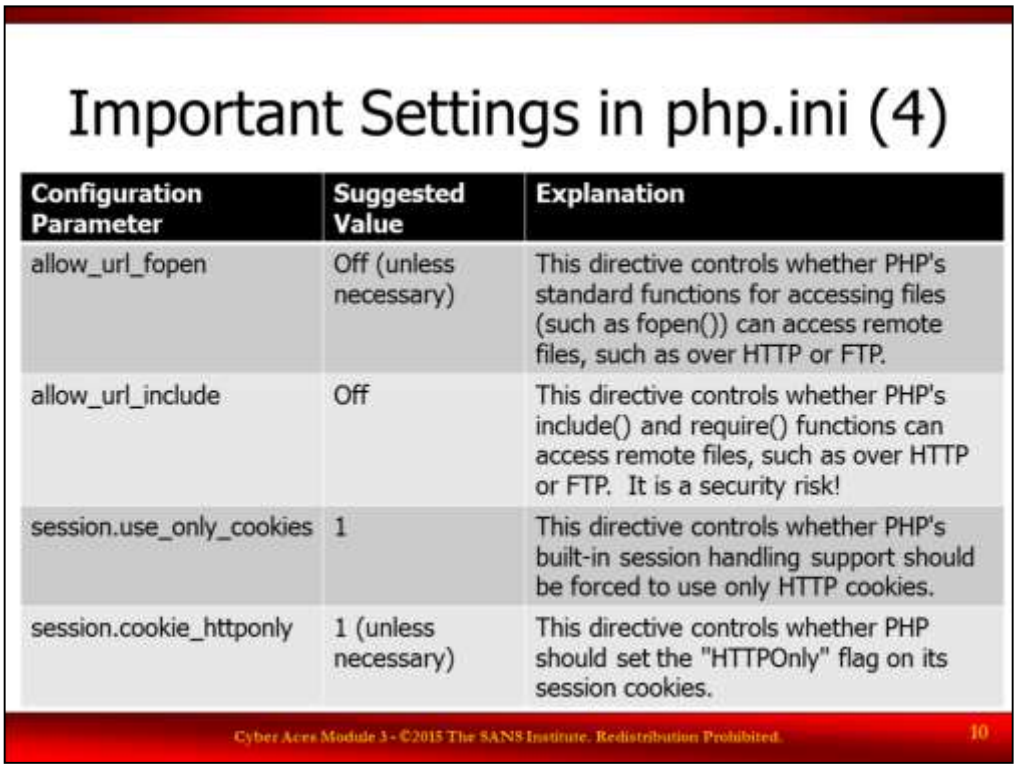
More detailed explanation of the above parameters:

max_execution_time (30): This directive controls the maximum amount of time that a PHP script may execute before it is terminated, specified in seconds. This should be tweaked depending on your environment, but 30 seconds is a reasonable default. Note that this directive does not apply to PHP scripts invoked from the command line, which have no time limit.

post_max_size (8M): This directive controls the maximum size that any POST request to the PHP script can be. Please note that if you need to handle file uploads, post_max_size must be larger than upload_max_filesize, because files are uploaded using POST. post_max_size controls how big an entire POST request may be, whereas upload_max_filesize controls how big a single uploaded file may be. This should be tweaked depending on your environment, but 8M is a reasonable default.

upload_max_filesize (2M):This directive controls the maximum size that any uploaded file to the PHP script can be. Please note that if you need to handle file uploads, post_max_size must be larger than upload_max_filesize, because files are uploaded using POST. This should be tweaked depending on your environment, but 2M is a reasonable default.

file_uploads (Off): This directive controls whether PHP is allowed to process file uploads (through web forms). Unless your site needs to handle file uploads, this feature should be turned off.



Important Settings in php.ini (4)

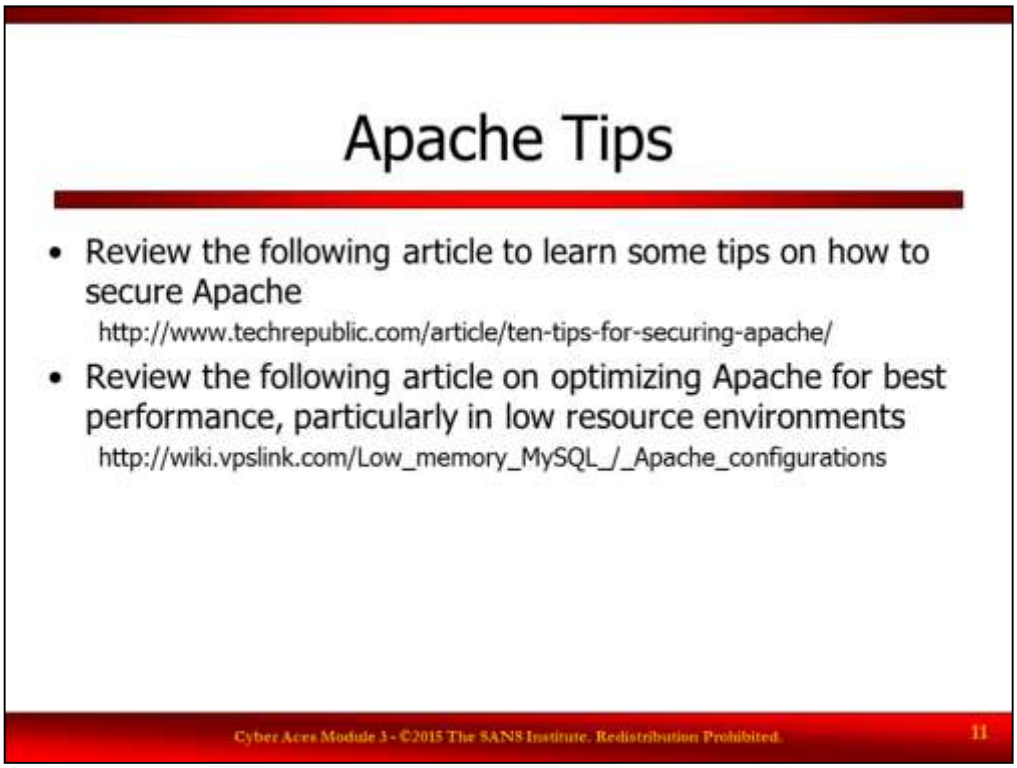
More detailed explanation of the above parameters:

allow_url_fopen (Off): This directive controls whether PHP's standard functions for accessing files (such as fopen()) can access remote files, such as over HTTP or FTP. This feature should be disabled unless its use is required. PHP's curl functions should be considered as an alternative.

allow_url_include (Off): This directive controls whether PHP's include() and require() functions can access remote files, such as over HTTP or FTP. This feature should always be disabled for security reasons, to defend against Remote File Include (RFI) attacks.

session.use_only_cookies (1): This directive controls whether PHP's built-in session handling support should be forced to use only HTTP cookies. It should always be enabled, as passing session data through URL's is insecure.

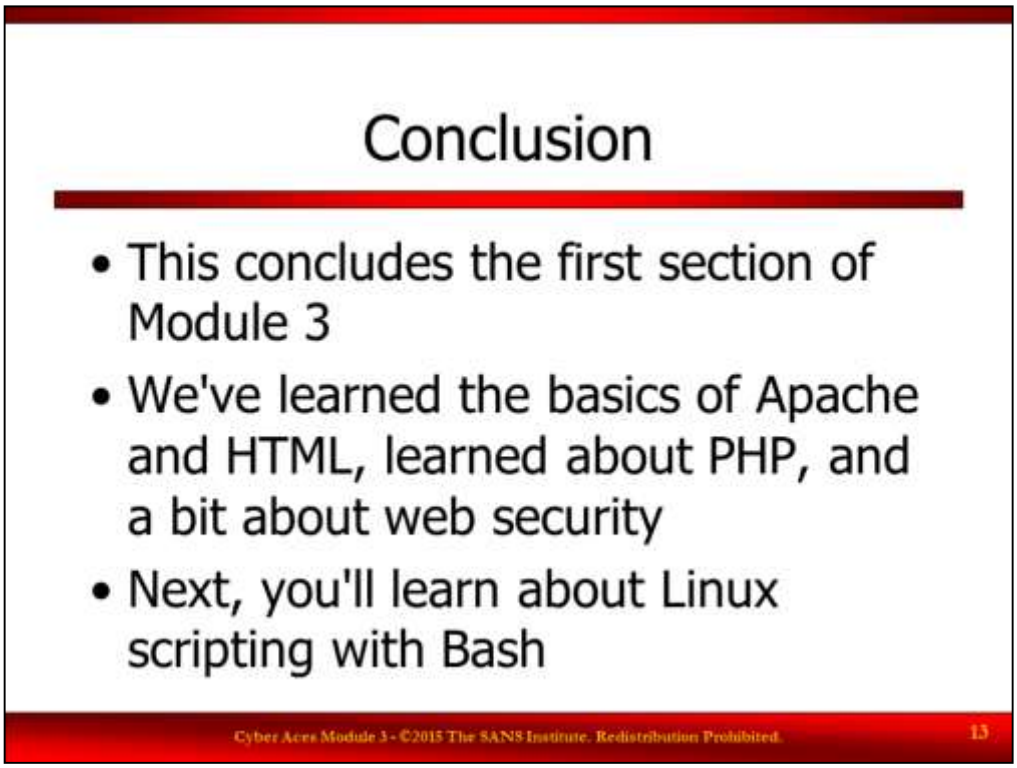
session.cookie_httponly (1): This directive controls whether PHP's built-in session handling support should set the "HTTPOnly" flag on its session cookies, a fairly new HTTP feature that prevents client-side JavaScript from reading cookie values. This is an important mitigation against Cross Site Scripting attacks (XSS) and should be enabled, unless you have a specific need for JavaScript to manipulate your session cookies.



Apache Tips

Apache can be configured through the httpd.conf file, but the many, many options it offers may seem confusing at first look. The most important configuration options you should worry about are the ones related to security. The guide at http://www.techrepublic.com/article/ten-tips-for-securing-apache/ goes over some tips on hardening your Apache installation against attackers. System administrators also make changes to the Apache configuration file to optimize the web server. Most of the default options are probably appropriate for your system and should not be changed for the sake of efficient resource usage; however, some may benefit from tweaking a few options such as StartServers and MaxClients to minimize memory usage under low-resource environments. The values of these options should be derived from the available resources of your system and number of visitors you expect to view your site. For most low-resource setups, following the highly optimized configuration at http://wiki.vpslink.com/Low_memory_MySQL_/_Apache_configurations is a good place to start (scroll down to the section on Apache).

This concludes the first section of Module 3, PHP and basic web security.



This concludes the first section of Module 3. We've learned the basics of Apache and HTML, and we've learned about PHP. We also touched on the basics of web security, discussing some of the most common attack vectors.

Next, you'll learn about Linux scripting with Bash, followed by Windows scripting with PowerShell!