Global Information Assurance Certification Paper

Configuring PHP for success and security

Eric Marshall

August 1st, 2004

Abstract: The success of the World Wide Web has led to the widespread use of dynamic web pages and attendant security problems. This paper describes a number of approaches and configuration options needed to use PHP to create and secure dynamic web pages.

Many organizations have connections to the Internet and have websites to mark their presence on the Web. Many of these organizations cannot afford the luxury of non-dynamic web sites as suggested by many IT security experts. Dynamic websites offer many advantages over other forms of software such as easy deployment to large numbers of users, platform independence, single point of maintenance, etc. while at the same time posing substantial security risks.

A potential problem in securing web sites, for many in IT security, is the security officer's lack of experience in web programming, the web developer's lack of experience in security and the dearth of material focused on this topic coupled with the wide range of technologies associated with the Web. In this paper, I will focus on the PHP language, a technology widely associated with the World Wide Web. I will look at problems associated with the configuration of PHP for use in web programming. I hope to provide an understanding of the major problems associated with web programming and a list of easy to spot items in and associated with PHP's configuration that will allow a security person to look at a configuration file and see if the obvious and reasonable first steps for security are being taken.

One of problems, for security officers, due to the wide range of material, in areas the security officer is not expert, is he or she is forced to trust the opinions of others within his or her organization in order to achieve true security for the organization. It critical to the success of the security officer that there is enough information to judge how much trust to place in programmers and system administrators, who create or manage parts of the perimeter that are notorious for security problems.

Brief Background

This paper focuses on the PHP scripting language and its use for web based applications and dynamic web pages. It is assumed that information in this paper will be seen as a small part of a larger context of best security practices and best software development practices. It is easy to not see the forest for the trees and just as easy to get lost among the trees for not seeing the forest regarding the process of writing, securing, and managing secure software.

Browser and web servers manage the flow of information whenever someone views a web page. When a web page is requested the browser sends that request via the Hypertext Transfer Protocol (HTTP)[i] at the application level. Communications takes place over the Transmission Control Protocol[ii]/Internet Protocol[iii] (TCP/IP).[iv] HTTP could be implemented on top of any web-based protocol[v] however. The web server, by convention, 'listens' on port 80 for non-

secure communications[vi] via HTTP and 'listens' to port 443 for secured communications[vii] via HTTPS (generally using Secure Sockets Layer (SSL) or Transport Layer Security (TLS)). Encryption of web-based communications is the topic of a paper itself and will not be covered here. For the level of this discussion, think of it as black box working at a lower layer of a connection, which otherwise works the same as HTTP. The web server, having received and parsed the request finds the appropriate file and sends it back to the browser[viii]. After receiving the requested information, the browser parses Hypertext Markup Language (HTML)[ix] or the Extensible HyperText Markup Language (XHTML) and displays a page, based on the instructions in the HTML, that the user then can see and read (and mostly likely repeat the process). The page of HTML may also have Cascading Style Sheets (CSS)[x] embedded within, which are a separate set of instructions affecting the layout, look and feel of a page. HTML and CSS are both stored as text. A page of HTML may also contain requests for other pieces of information to be fetched via HTTP. Most common being digital images stored in formats like .jpg, .gif, .png, .bmp, etc. The browser manages all of these types of media and places them as they are received in the rendered page.

The limitation of this model is the static nature of the requested web pages. Dynamic web pages overcome this limitation allowing the creators of these web pages to display near-real time data like weather conditions or stock quotes and alter the page on the fly. Dynamic pages use the same protocols of HTTP and HTML. The difference is in the web server and/or in the browser.

Most browsers support one or more similar languages: JavaScript, VBScript or ECMAScript. JavaScript[xi] was created by Netscape (and bears no relationship to Java, other than in name), VBScript[xii] was developed by Microsoft in response to JavaScript and ECMAScript[xiii] is openly standardized (by European Association for Standardizing Information and Communication Systems) version of JavaScript. Each browser supports one or more versions of these languages, which allows a web programmer to run code on the user's browser. A simple example would be to store a visitor's preferences in a cookie[xiv] (a small text file stored and managed by the visitor's web browser). When the visitor returns to the site, script embedded in the web page will execute and read the preferences and alter the web page's structure to match the preferences. Another common use of browser-side scripts is pop-up windows informing the visitor of an incorrectly filled-in form. As an aside, web developers should always assume that all browser-side scripting could be bypassed or altered. Browser side scripting is very powerful mechanism for controlling most aspects of the browser. However, most (if not all) browsers capable of running these kinds of code, can be configured to not execute any such code.

Dynamic web pages can also be constructed on the server side before they are sent to the browser. When the web server receives a request for a page, instead of finding the page of HTML text and sending it back, the web server can do one of two other things: one, execute the file as program, collect the output and send

that back along or two, read the file before sending and wherever the web server encounters special commands to execute snippets of code, the web server executes the code and replace the special commands with the output of the executed code and then returns the reworked page. Depending on the site either one or both approaches may be used.

If the file is executed and the web server waits for the output; this is called, a Common Gateway Interface (CGI)[xv] script or program. CGI is a specification for this approach (which supports a very wide range of programming languages). PHP can be used in this way. The web server must be configured to so it 'knows' to execute PHP scripts. Apache can be configured (as most web servers) to execute a file based on the file suffix, in this case .php.[xvi] Usually a directory is specified (for example, www.sample.org/cgi-bin/myscript) as well as a suffix is identified like .php or .cgi (for example, www.sample.org/todays_results.php or www.sample.org/todays_results.cgi). In the first line of a CGI PHP script needs to identify where the php executable is that parses and executes the script. This will generally look this `#!/usr/local/bin/php` but this path is system dependent.

The other approach works like Server Side Includes[xvii] (supported by Apache[xviii] and NCSA HTTPd[xix] web servers) or Active Sever Pages[xx], where the web server scans for special tags, in the HTML text that require processing to be done by the server. The results are given back to the web server where the special tags are replaced by the results allowing web developers to add the PHP code equivalent of <put today's date here>, which when viewed show the current date. PHP's special tags/commands begin with `<?php` and end with ?>. For example please see Source Listing 1 (as found in the Appendix of this paper). The line `<?php echo '<p>Hello World</p>'; ?>` would be replaced with `<p>Hello World</p>` While this not very useful, inserting the current time or information stored in a database is almost as easy and is very useful. When PHP is used this way, for any changes made in PHP's configuration file to take effect, the web server must be restarted.[xxi]

PHP can be used in a third way as a command line script. The command line form is not used to serve web pages and will not be discussed here.

PHP runs on all major platforms and can be used with most web servers, including Apache, Microsoft IIS and Netscape/iPlanet servers. While the language can be used in a variety of ways, PHP is primarily associated with dynamic web content.

Rasmus Lerdorf created PHP[xxii] in 1994 and since then PHP's usefulness, portability and user base have only grown. PHP has grown and adapted to it large and diverse user base by expanding the range of its functionality. Since PHP has a reputation for being easy to use, it poses an extra rich possibility for security concerns since easy-to-use languages attract a larger number of non-

programmers (or what one may call semi-programmers) who learn and use the language a little bit at a time and who may not be aware or concerned about issues of security. Highly trained programmers also may not be aware or concerned since not all organizations train developers about security nor is security a focal point of university or college training in computer science and related fields yet.

This paper will not cover web server configuration and security nor server OS hardening and the like which are obviously necessary parts of a Defense in Depth approach needed to protect an organization that servers dynamic pages on the Web.

Configuration

Configuration is handled via the php.ini file (or php3.ini for the older version of PHP, PHP3). PHP does not support other names for the configuration file. This file is read when the web server starts. If no php.ini is found, then PHP will load with its own pre-set defaults. For Common Gateway Interface (CGI) scripts, php.ini is read on every invocation.[xxiii] On most Linux and Unix installs this will reside in the /etc/ directory, on Mac OSX, php.ini will be found at `/usr/local/bin/php.ini` and on Windows 9x/ME/XP php.ini will be found in `%WINDIR%` which is usually C:\Windows. On Windows NT/2000 will be in `%WINDIR%` or `%SYSTEMROOT%` that is usually C:\WINNT of C:\WINNT40 for NT/2000 servers.[xxiv] Please note that the php.ini can be placed elsewhere at the discretion of the system administrator. One source noted if PHP did not find the php.ini file it did not complain, so be aware of this when troubleshooting problems.[xxv] (There is display_startup_errors option that might catch this problem.)

Comments are supported; everything after a semi-colon is ignored. All directives use the same syntax: option = value. (Please note that period after 'value' merely finishes that sentence and is not part of the syntax.) Values can be a string, a number, a PHP constant (e.g. `E_ALL` or `M_PI`), one of the INI constants (`On`, `Off`, `True`, `False`, `Yes`, `No` and None) or an expression (e.g. `E_ALL & ~E_NOTICE`), or a quoted string (`"foo"`). Expressions in the php.ini file are limited to bitwise operators and parentheses: | (bitwise OR), & (bitwise AND), ~ (bitwise NOT) and ! (Boolean NOT). Boolean flags can be turned on using the values 1, On, True or Yes. They can be turned off using the values 0, Off, False or No. [xxvi] To denote an empty string, the value 'none' (with out quotation marks) can be used or simply have no value to the right of the equals mark (like so option =).

register_global

An option that needs to be configured as 'Off' is register_globals[xxvii]. If this set to 'On', PHP will automatically take input sent as part of a HTML URL or posted

data from an HTML form and create a variable with an assigned value. For example, if my web page has a form asking for first name and last name it would look like Image 1 (as found in the Appendix of this paper). The HTML code would look the code found in Source Code Listing 2 (as found in the Appendix of this paper). If register_globals is on, then in the PHP script myniftyPHP.php, there will be two variables created: first_name and last_name, each initialized with the value of what was typed in the respective textboxes. So if a user types in 'Jane' and 'Doe' for first name and last name then in myniftyPHP.php, there will be two variables in memory: first_name='Jane' and last_name='Doe'. At first glance this may seem like a convenience, but as we will see this is not. This 'convenience' allows a malevolent user to create any variable one likes by altering the URL as easily as `http://www.ourwebsite.com/myniftyPHP.php?trusted=true`

Now myniftyPHP.php starts with a variable named `trusted` equal to true. This allows the malevolent user to create values for any variable in the PHP program.

To avoid this, make sure in php.ini there is this line:

```
register_globals = Off
```

safe_mode

safe_mode prevents users from accessing information and files belonging to other users. When using any file operation, through functions such as `copy()`, `chmod()`, `chown()`, `chgrp()`, `dir()`, `file()`, `flock()`, `fopen()`, `mkdir()`, `move()`, `remove()`, `readfile`, `rmdir()`, and `unlink()` among others, PHP checks the file permissions of script running against the file or directory being accessed. If the permissions don't match, the operation is prevented. To engage safe_mode set:

```
safe_mode = On
```

The functions like `include()` and `require()` are likewise checked but the files accessed must also be within the directory specified by the configuration option `safe_mode_include_dir` in php.ini or the action will fail.

Functions like exec(), passthru() and system() are limited by safe_mode to executables found in `safe_mode_exec_dir` as set in php.ini

By setting `safe_mode_allowed_env_vars`, scripts can only manipulate environmental variables starting with prefixes listed (comma delimited). By default, `safe_mode_allowed_env_vars` set to `PHP_` .

The PHP manual lists these functions as altered by safe_mode: `dbmopen()`, `dbase_open()` ,`filepro()`, `filepro_rowcount()`, `filepro_retrieve()`, `pg_lo_import()`, `posix_mkfifo()`, `putenv()`, `move_uploaded_file()`, `chdir()`, `dl()`, `shell_exec()`, `exec()`, `system()`, `passthru()`, `popen()`, `fopen()`, `mkdir()`, `rmdir()`, `rename()`, `unlink()`, `copy()`,

`chgrp()`, `chown()`, `chmod()`, `touch()`, `symlink()`, `link()`, `apache_request_headers()`, `header()`, `highlight_file()`, `show_source()`, `parse_ini_file()`, and `mail()`.[xxviii] Please note however, the PHP manual adds this caveat regarding the influence of safe_mode: “This is a still probably incomplete and possibly incorrect listing of the functions limited by safe mode.”[xxix]

open_basedir

If `open_basedir` is set, all file operations are limited to the specified directory and the subdirectories contained within. This option is independent of how safe_mode is set. This minimizes the range of remote file attacks to get at password files, configuration files and the like. If not used this option is commented out, if used it would look this (depending on local setup):

```
open_basedir = /var/www/html/
```

error_reporting

The error_reporting directive takes a series of bit flags that are logically OR’ed or logically ANDed together. The flags are defined in the php.ini file as follows[xxx]:

E_ALL - All errors and warnings
E_ERROR - fatal run-time errors
E_WARNING - run-time warnings (non-fatal errors)
E_PARSE - compile-time parse errors
E_NOTICE - run-time notices (these are warnings which often result from a bug in your code, but it's possible that it was intentional (e.g., using an un-initialized variable and relying on the fact it's automatically initialized to an empty string)
E_STRICT - run-time notices, enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code
E_CORE_ERROR - fatal errors that occur during PHP's initial startup
E_CORE_WARNING - warnings (non-fatal errors) that occur during PHP's initial startup
E_COMPILE_ERROR - fatal compile-time errors
E_COMPILE_WARNING - compile-time warnings (non-fatal errors)
E_USER_ERROR - user-generated error message
E_USER_WARNING - user-generated warning message
E_USER_NOTICE - user-generated notice message

The following operators work: | (bitwise OR), & (bitwise AND), and ~ (bitwise NOT) so to see all errors except strict notices the following would be used: E_ALL & ~E_STRICT.
I recommend logging all errors, which avoids the use of bit-wise operators.

```
error_reporting  = E_ALL
```

html_errors
This will suppress HTML warnings and errors if turned on. The php.ini file has the following comment regarding this – "Note: Never use this feature for production boxes".[xxxi]

```
html_errors = Off
```

ignore_repeated_errors
ignore_repeated_source
PHP can ignore repeated errors. I believe there is no benefit in this feature's use, since my approach is to catch all errors and then safeguard against each error found. If this is not your approach and space for logging is limited, this option exists. Ignore_repeated_source will ignore repeated errors from different line numbers or different files.

```
ignore_repeated_errors = Off
ignore_repeated_source = Off
```

display_errors
display_errors should be set to off for production servers. There is no need to give any extra information like file paths or database information to someone trying to see how things break on your system. This should be set to 'on' only on development servers but keep in mind when debugging failures in production that this directive suppresses errors so developers might be 'thrown off the scent' when first trying hunt down problems.

```
display_errors = Off
```

log_errors
This directive allows PHP to write to a log file. This is most useful in catching problems and errors without passing information to users or attackers. This will hopefully redeem you with the developers you irked by turning off 'display_errors'.

```
log_errors = On
```

track_errors
This directive enables PHP to store the last error in the environmental variable `$php_errormsg`. If your applications have complex error handling, this may be required, otherwise if this environmental variable is not used then set this directive '`Off`'. It may be wise to check with your developers regarding this. (It would also be wise to document any such dependencies.)

```
track_errors = Off
```

error_log

All errors deems noteworthy (as based on the `error_reporting` directive) will be logged to a specified file. `error_log` also supports reporting via syslog[xxxii]. Beware if using syslog of the `define_syslog_variables` directive.

```
error_log = /var/log/httpd/php_error.log
```
or
```
error_log = syslog
```

allow_url_fopen
By default `allow_url_fopen` lets PHP treat URLs as files. Very few sites require remote files functionality.[xxxiii] It should be set as so:

```
allow_url_fopen = Off
```

In older versions of PHP (pre-4.0.3) this is set as a compiler option.

disable_functions
This disables all comma-delimited listed functions from working. This very useful for limiting PHP's access to the file system and operating system by removing functions like `system()` or `shell_exec()`. Another function worth disabling is `phpinfo()` since it displays all configuration information.

disable_classes
`disable_classes` works the same as `disable_functions` but for classes instead of functions. (Classes are an object-oriented feature of PHP and at the risk of over-simplifying a complex idea, can be thought as functions bundled with their own variables.)

expose_php
This option decides if PHP exposes the fact it is running on the server. While there is no direct security risk by setting this to 'on', setting to 'off' may make it a little harder for an attacker to understand how your website is set up.

```
expose_php = Off
```

file_upload
If one does not require uploading files then disable this. There is no good that can come from having attackers upload files in order to attack your system more efficiently.

```
file_upload = Off
```

upload_max_filesize
If one does need to upload files then this sets the maximum file size that can be uploaded through PHP. The size is in bytes.

```
upload_max_filesize = 2097152
```

max_execution_time
This option limits the amount time a PHP script can run for, in seconds. The default is thirty seconds. Lower is better, but if set too low, your scripts will not run.

```
max_execution_time = 30
```

max_input_time
This caps the maximum amount of time each script may spend parsing requests, in seconds. Again, shorter is better. The default is sixty seconds.

```
max_input_time = 60
```

memory_limit
This limits the amount of memory a script can use. The default is eight megabytes.

```
memory_limit = 8M
```

variables_order
variables_order set the order in which PHP registers variables from GET, POST, cookies, environment and built-in variables. This is important in cases where namespaces of variables overlap. This will protect against attackers trying to exploit oversights in variable names or assumptions about how certain pieces of information are created or where they came from. Variables are handled from left to right with newer values overriding older values.

```
variables_order = "EGPCS"
```

E = environmental variable
G = GET
P = POST
C = cookie
S = system built-in variable

y2k_compliance
This will cause problems with non-compliant browsers.
```
y2k_compliance = On
```

magic quotes
`magic_quotes_gpc` creates backslashes to escape all ' (single-quote), " (double quote), \ (backslash) and NUL's. This is a problematic setting. While doing mostly good, there are a few problems associated with setting this on.

There is not space here to describe all the details of SQL injection attacks, other than to say that like problems with `system()` and strings, it is equally easy to add characters to the end of a SQL query so commands can be passed to the database management system (and some database management systems have commands similar to `system()`). If these directives are turned off, then each application must manage string handling for any database queries and the like manually.

```
magic_quotes_gpc = Off
magic_quotes_runtime = Off
magic_quotes_sybase = Off
```

arg_separator.input and arg_separator.output
These directives manage how PHP parses URLs. Shown below are the default values.

```
arg_separator.input = ";&"
arg_separator.output = "&amp;"
```

doc_root, cgi.force_redirect and cgi.redirect_status_env
If PHP is being used for CGI, it is critical to configure PHP so the PHP interpreter is not accessible by attackers via the CGI mechanism. Please consult the PHP manual for more detail (http://us3.php.net/security.cgi-bin) and CERT's Advisory CA-1996-11 - Interpreters in CGI bin Directories (http://www.cert.org/advisories/CA-1996-11.html). If doc_root is in conjunction with safe_mode, no files outside this directory are served. Please note for Microsoft's Internet Information Services[xxxiv], cgi.force_redirect must be turned off.

sql.safe_mode
I could find no information as what this directive does! The default value is off.

```
sql.safe_mode = Off
```

session directives
PHP supports a number of functions to "preserve certain data across subsequent accesses"[xxxv]. For more information please see http://us2.php.net/session.

SQL directives
The php.ini file contains a large number of directives for different database products. Each different database has its own set of functions and directives. For more information please see the master list of functions (http://us4.php.net/manual/en/) for details.

Best Practices

These recommendations are based on the Open Web Application Security Project's top ten security vulnerabilities.[xxxvi] This list is not aimed at PHP directly but at all web applications. Some of this material is also best practices for any software development (even before software best practices included security as a component).

Avoid buffer over-runs

PHP handles all dynamically allocated memory for the user. So an attacker cannot create a buffer over-run in variables used by the PHP developer. However, PHP is written in C, a language famous for buffer over-runs. This means it is important to keep track of patches and new releases in case a buffer over-run is found in PHP itself. Until operating systems (or middleware layers sitting on the operating systems) can detect problems with on the stack, these kinds of attacks will be found in code.[xxxvii] Diligent patching is the only real defense (unless you have vast resources to comb through the source code to find these kinds of errors yourself and patch them).

Avoid the shell

PHP contains a number of functions that will pass a string to the operating system to execute. While this convenient, it is an infamous path of exploitation when trying to hack a system. If a malevolent user can figure out how to modify the string passed to the system, he or she can execute any command that PHP can. By simply adding ";" to the end of the string and then add a command like "`rm –r *` " the operating system on most UNIX systems will execute two commands: the one the PHP script intended and the second, "`rm –r *`". Do not use `shell_exec(), exec(), system(), passthru(), popen()` or back ticks (`) in one's PHP scripts. Using disable_functions in the configuration file to suppress these functions can enforce this policy.

If one must use calls like `system(),` filter all data from the outside (user input, cookies, environmental variables, etc.) and do not use the data directly. Instead of passing a string via variable, use the variable in an if-statement to trigger a hard coded command. If this isn't possible, it is still possible to assemble a string from a series of if-statements so no user data is passed directly to the `system()` but instead merely triggers a concatenation of pre-stored strings. This is a very powerful and easy technique to use to side-step very nasty problems. Assume the shell cannot be secured no matter how hardened the operating system is.

Check that code/code libraries and data are not in the web server's document root

The actual PHP engine and its libraries should not be visible to the web. Make sure that these files are not in the web server's document root. Otherwise, malevolent viewers will be able to create and exploit weakness much easier. For PHP code that has information that ought be best keep secure like database scripts that contain passwords (but, of course, that style of programming should be avoided) it is possible to create custom functions that are outside of the web server's document root. This is still possible however to compromise this information in custom functions.

Check the permissions of the script
Check the permissions of the web server
Check the permissions of the data

PHP scripts should not run as root (or any other equally powerful user) otherwise a simple coding mistake would comprise the entire server. The web server should not execute PHP scripts as root nor should scripts or data used by PHP be executed as root. Apache handles this by using the user 'nobody' in order to not use 'root'. (Part of Apache does run as root in order to create sockets for the correct ports. This part is a separate fork/thread from the rest of Apache that runs as 'nobody')
If safe_mode is used, this will be checked automatically. Thus preventing malevolent users from accessing things they should not.

Don't show errors, log errors

When attacking a server, the first stage is collecting information about the server and its applications. Error messages are a valuable resource to an attacker, telling him or her about software packages, version numbers and sources of errors (which can be exploited directly). So it is in the web master's direct interest to suppress this information for outside users and limit an attacker's ability to explore. It is also in the web master's direct interest to be aware of any errors so they can be patched, fixed and handled as quickly as possible, so logging of PHP errors is important as well. This is handled in php.ini by `display_errors` and `log_errors` as well as a few other options that handle the location of the log file and the like. Please note: on development servers, these options should not be set since it makes debugging much harder and thus slows development.

Don't trust any user input

A core precept in web application security is never trust user input. All information coming from the outside should be considered tainted at best. PHP does not provide a taint flag like Perl does, so there is no easy way to check if input is handled correctly. This responsibility rests on the programmer.

All information coming from user input should be suspect. This includes variables from hidden fields[xxxviii], cookies, environmental variables and all user input. While it might seem unlikely, all of these sources can be easily overcome with simple tools like a text editor. As mentioned above, sometimes using user input can be avoided or sidestepped. When it cannot, all data must be scrubbed.

Cross-site scripting exploits are the result of unscrubbed data being posted on a web page. As an example, Randal Schwartz (of Perl fame) has the following on his home page: “One of my former "stupid Randal tricks" has been to cruise the net for guest books and see if they accept raw HTML, testing it by feeding it a name or comment of Barney `< IMG SRC = http://barneyonline.com/Barney/Images/Home/iconBarney1.jpg >`. It's amazing how many of them blindly accept it. (I can't stop giggling when I see the Purple One show up in the list when I reload.)”[xxxix] The defaced guest book will display a picture of Barney (or a broken link to a former picture of Barney). This technique applies to inserting code into programs parsing input data as well resulting in more dangerous results.

Many non-alpha-numeric characters have special meanings for different operating systems, languages, databases and the like. Replace non-alpha-numeric characters with something harmless. Keep in mind that an ampersand and a number can specify any character so all the characters you thought you scrubbed away might have still sneaked past (pardon the mixed metaphor). There are different levels of encoding and different regional encodings as well: all of which may be handled differently depending on which web server is used. http://www.asciitable.com/ has a delightful chart showing 640 ways to encode the 128 characters of American Standard Code for Information Interchange (ACSII)[xl] There are also issues between ACSII/Standard ECMA-6[xli] and Unicode[xlii] conversions as well with mapping Microsoft’s Windows Latin-1a superset of ISO 8859-1, onto Unicode correctly. This is just the tip of the iceberg.

Scrubbing input is a most important area of diligence. If only one message in this paper stays with you, the reader let it be “Do not trust any input”. I personally have found issues like this with national online booksellers and government sites including NASA. Please read CERT Advisory CA-1997-25 Sanitizing User-Supplied Data in CGI Scripts (http://www.cert.org/advisories/CA-1997-25.html), CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests (http://www.cert.org/advisories/CA-2000-02.html) and CERT Coordination Center’s How To Remove Meta-characters From User-Supplied Data In CGI Scripts (http://www.cert.org/tech_tips/cgi_metacharacters.html) for more information.

Initialize all variables

It is easy to overlook initializing variables[xliii]. This can cause hard to track bugs in languages like C and C++, unexpected behavior in complex data structures in

Perl and Java and may allow attackers to alter code behavior of web applications. In Source Listing 3 (found in the Appendix), there is no problem if `good_to_go` is defined as false somewhere at the beginning of the program. However, if this step is bypassed and register_globals has it default value of '`On`', an attacker can define `good_to_go` as true in the URL and the attacker will gain access to the password-protected content.

Hide important data

Important data should always be stored in a separate file, away from the script that uses it. The data should be stored in a directory that cannot be accessed via a web server request. When the data in question is needed, that data can be included in the PHP script via `include()` or `require()` function calls.

Conclusion

While securing dynamic websites is a complex and challenging task, the demand for these types of sites only grows. Dynamic web sites will only grow more prevalent as time passes. Organizations like SANS will eventually rise to task to help train the next generation of web administrators and web developers to create secure web-based applications. To create secure web-based applications requires knowledge of HTML, web servers, programming languages, a bit of networking, a bit of system administration, deep knowledge of security issues and in many cases, knowledge of other areas like databases, interface design, graphics design and CSS. Creating software of this complexity is a group effort and requires communication and understanding of the role security plays by all contributors. PHP is an effective and powerful tool. As with all powerful tools, however, PHP must be set-up carefully to insure its safe use. Configuration is only the beginning of the safe use of PHP. Hopefully this paper has touched on some of these issues without over-whelming the reader.

Bibliography:

[1] Apache Software Foundation "Security Tips for Server Configuration". URL: http://httpd.apache.org/docs/misc/security_tips.html (3 July 2004)

[2] National Center for Supercomputing Applications Software Development Group. "Server Side Includes (SSI)". 28th September 1995. URL: http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html (3 July 2004)

[3] Bakken, Stig Sæther et al. PHP Manual. 9th June 2004. URL: http://www.php.net/manual/en/index.php (3 July 2004)

[4] Berners-Lee, T. "RFC 1945 – Hypertext Transfer Protocol – http/1.0". May 1996. URL: http://www.faqs.org/rfcs/rfc1945.html (3 July 2004)

[5] Bos, Bert, Çelik, Tantek, Hickson, Ian And Lie, Håkon Wium. "Cascading Style Sheets, level 2 revision 1 CSS 2.1 Specification". 25th February 2004. URL: http://www.w3.org/TR/CSS21/

[6] Brogdon, Darrell. 29 "Securing a PHP Installation". March 2001. URL: http://www.onlamp.com/pub/a/php/2001/03/29/php_admin.html (3 July 2004)

[7] Coar, K. "The Common Gateway Interface - RFC Project Page ". URL: http://cgi-spec.golux.com/ (3 July 2004)

[8] Clowes, Shaun "A Study In Scarlet, Exploiting Common Vulnerabilities in PHP Applications". URL: http://www.secureReality.com.au/archives/studyinscarlet.txt (3 July 2004)

[9] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. June 1999 "Hypertext Transfer Protocol -- HTTP/1.1". ftp://ftp.isi.edu/in-notes/rfc2616.txt (3 July 2004)

[10] Fuecks, Harry. "register_globals off". The PHP Anthology Volume 1, Chapter 1 – PHP Basics. 19th December 2003. URL: http://www.sitepoint.com/article/php-anthology-1-1-php-basics/6 (3 July 2004)

[11] Lonvick, C. "RFC 3164 - The BSD Syslog Protocol". August 2001 URL: http://www.faqs.org/rfcs/rfc3164.html (3 July 2004)

[12] Open Web Application Security Project. "The Ten Most Critical Web Application Security Vulnerabilities". 13th January 2003. URL: http://www.owasp.org/documentation/topten (3 July 2004)

[13] Pemberton, Steve et al. "XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)". 1st August 2004. URL: http://www.w3.org/TR/xhtml1/ (3 July 2004)

[14] Raggett, Dave. Le Hors, Arnaud. and Jacobs, Ian. "HTML 4.0 Specification" 24th April 1998. URL: http://www.w3c.org/MarkUp/ (3 July 2004)

[15] Rescorla, E. "RFC 2818 - HTTP Over TLS" May 2000. URL: http://www.faqs.org/rfcs/rfc2818.html (3 July 2004)

[16] Ristic, I. "PHP Configuration". Late 2003. URL: http://www.webkreator.com/php/configuration/php-configuration.html (3 July 2004)

[17] Robinson, David. ‘‘The CGI Specification". 16th October 1995. URL:http://hoohoo.ncsa.uiuc.edu/cgi/interface.html (3 July 2004)

[18] Sklar, David. "PHP and the OWASP Top Ten Security Vulnerabilities" URL: http://www.sklar.com/page/article/owasp-top-ten (3 July 2004)

[19] Stein, Lincoln and Stewart, John "CGI (Server) Scripts" The World Wide Web Security FAQ URL: http://www.w3.org/Security/Faq/wwwsf4.html (3 July 2004)

[20] Wheeler, David. "Secure Programming for Linux and Unix HOWTO". March 2003. URL: http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html (3 July 2004)

[21] Youman, Yves. "An overview of common programming security vulnerabilities and possible solutions". August 2003. URL: http://fort-knox.org/thesis.php (3 July 2004)

Appendix:

Image 1:

Source Listing 1:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en">
  <head>
   <title>echo example</title>
  </head>
  <body>
  <?php echo '<p>Hello World</p>'; ?>
 </body>
 </html>
```

Source Listing 2:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en">
<head>
…
      <form action=”http://www.ourwebsite.com/myniftyPHP.php“>
            First Name:
            <input type=”text” name=”first_name”>

            Last Name:
            <input type=”text” name=”last_name”>
```

Source Listing 3:

```
<? php
      ...
      if ($user_password == $secret_password_from_database )
      {
            $good_to_go = 1;
      }
      ...
```

```
     if ($good_to_go  == 1)
     {
          do_something_only_users_with_passwords_can();
     }
?>
```

Default php.ini file from http://cvs.php.net/co.php/php-src/php.ini-dist:

```
[PHP]

;;;;;;;;;;
; WARNING ;
;;;;;;;;;;
; This is the default settings file for new PHP installations.
; By default, PHP installs itself with a configuration suitable for
; development purposes, and *NOT* for production purposes.
; For several security-oriented considerations that should be taken
; before going online with your site, please consult php.ini-recommended
; and http://php.net/manual/en/security.php.


;;;;;;;;;;;;;;;;;;;
; About php.ini   ;
;;;;;;;;;;;;;;;;;;;
; This file controls many aspects of PHP's behavior.  In order for PHP to
; read it, it must be named 'php.ini'.  PHP looks for it in the current
; working directory, in the path designated by the environment variable
; PHPRC, and in the path that was defined in compile time (in that order).
; Under Windows, the compile-time path is the Windows directory.  The
; path in which the php.ini file is looked for can be overridden using
; the -c argument in command line mode.
;
; The syntax of the file is extremely simple.  Whitespace and Lines
; beginning with a semicolon are silently ignored (as you probably guessed).
; Section headers (e.g. [Foo]) are also silently ignored, even though
; they might mean something in the future.
;
; Directives are specified using the following syntax:
; directive = value
; Directive names are *case sensitive* - foo=bar is different from FOO=bar.
;
; The value can be a string, a number, a PHP constant (e.g. E_ALL or M_PI), one
; of the INI constants (On, Off, True, False, Yes, No and None) or an expression
; (e.g. E_ALL & ~E_NOTICE), or a quoted string ("foo").
;
; Expressions in the INI file are limited to bitwise operators and parentheses:
; |       bitwise OR
; &       bitwise AND
; ~       bitwise NOT
; !       boolean NOT
;
; Boolean flags can be turned on using the values 1, On, True or Yes.
; They can be turned off using the values 0, Off, False or No.
```

```
;
; An empty string can be denoted by simply not writing anything after the equal
; sign, or by using the None keyword:
;
;  foo =         ; sets foo to an empty string
;  foo = none    ; sets foo to an empty string
;  foo = "none"  ; sets foo to the string 'none'
;
; If you use constants in your value, and these constants belong to a
; dynamically loaded extension (either a PHP extension or a Zend extension),
; you may only use these constants *after* the line that loads the extension.
;
;
;;;;;;;;;;;;;;;;;;;
; About this file ;
;;;;;;;;;;;;;;;;;;;
; All the values in the php.ini-dist file correspond to the builtin
; defaults (that is, if no php.ini is used, or if you delete these lines,
; the builtin defaults will be identical).


;;;;;;;;;;;;;;;;;;;
; Language Options ;
;;;;;;;;;;;;;;;;;;;

; Enable the PHP scripting language engine under Apache.
engine = On

; Enable compatibility mode with Zend Engine 1 (PHP 4.x)
zend.ze1_compatibility_mode = Off

; Allow the <? tag.  Otherwise, only <?php and <script> tags are recognized.
; NOTE: Using short tags should be avoided when developing applications or
; libraries that are meant for redistribution, or deployment on PHP
; servers which are not under your control, because short tags may not
; be supported on the target server. For portable, redistributable code,
; be sure not to use short tags.
short_open_tag = On

; Allow ASP-style <% %> tags.
asp_tags = Off

; The number of significant digits displayed in floating point numbers.
precision    =  12
```

```
; Enforce year 2000 compliance (will cause problems with non-compliant
browsers)
y2k_compliance = On

; Output buffering allows you to send header lines (including cookies) even
; after you send body content, at the price of slowing PHP's output layer a
; bit.  You can enable output buffering during runtime by calling the output
; buffering functions.  You can also enable output buffering for all files by
; setting this directive to On.  If you wish to limit the size of the buffer
; to a certain size - you can use a maximum number of bytes instead of 'On', as
; a value for this directive (e.g., output_buffering=4096).
output_buffering = Off

; You can redirect all of the output of your scripts to a function.  For
; example, if you set output_handler to "mb_output_handler", character
; encoding will be transparently converted to the specified encoding.
; Setting any output handler automatically turns on output buffering.
; Note: People who wrote portable scripts should not depend on this ini
;       directive. Instead, explicitly set the output handler using ob_start().
;       Using this ini directive may cause problems unless you know what script
;       is doing.
; Note: You cannot use both "mb_output_handler" with "ob_iconv_handler"
;       and you cannot use both "ob_gzhandler" and "zlib.output_compression".
; Note: output_handler must be empty if this is set 'On' !!!!
;       Instead you must use zlib.output_handler.
;output_handler =

; Transparent output compression using the zlib library
; Valid values for this option are 'off', 'on', or a specific buffer size
; to be used for compression (default is 4KB)
; Note: Resulting chunk size may vary due to nature of compression. PHP
;       outputs chunks that are few hundreds bytes each as a result of
;       compression. If you prefer a larger chunk size for better
;       performance, enable output_buffering in addition.
; Note: You need to use zlib.output_handler instead of the standard
;       output_handler, or otherwise the output will be corrupted.
zlib.output_compression = Off

; You cannot specify additional output handlers if zlib.output_compression
; is activated here. This setting does the same as output_handler but in
; a different order.
;zlib.output_handler =

; Implicit flush tells PHP to tell the output layer to flush itself
; automatically after every output block.  This is equivalent to calling the
; PHP function flush() after each and every call to print() or echo() and each
```

```
; and every HTML block.  Turning this option on has serious performance
; implications and is generally recommended for debugging purposes only.
implicit_flush = Off

; The unserialize callback function will called (with the undefind class'
; name as parameter), if the unserializer finds an undefined class
; which should be instanciated.
; A warning appears if the specified function is not defined, or if the
; function doesn't include/implement the missing class.
; So only set this entry, if you really want to implement such a
; callback-function.
unserialize_callback_func=

; When floats & doubles are serialized store serialize_precision significant
; digits after the floating point. The default value ensures that when floats
; are decoded with unserialize, the data will remain the same.
serialize_precision = 100

; Whether to enable the ability to force arguments to be passed by reference
; at function call time.  This method is deprecated and is likely to be
; unsupported in future versions of PHP/Zend.  The encouraged method of
; specifying which arguments should be passed by reference is in the function
; declaration.  You're encouraged to try and turn this option Off and make
; sure your scripts work properly with it in order to ensure they will work
; with future versions of the language (you will receive a warning each time
; you use this feature, and the argument will be passed by value instead of by
; reference).
allow_call_time_pass_reference = On

;
; Safe Mode
;
safe_mode = Off

; By default, Safe Mode does a UID compare check when
; opening files. If you want to relax this to a GID compare,
; then turn on safe_mode_gid.
safe_mode_gid = Off

; When safe_mode is on, UID/GID checks are bypassed when
; including files from this directory and its subdirectories.
; (directory must also be in include_path or full path must
; be used when including)
safe_mode_include_dir =

; When safe_mode is on, only executables located in the safe_mode_exec_dir
```

```
; will be allowed to be executed via the exec family of functions.
safe_mode_exec_dir =

; Setting certain environment variables may be a potential security breach.
; This directive contains a comma-delimited list of prefixes.  In Safe Mode,
; the user may only alter environment variables whose names begin with the
; prefixes supplied here.  By default, users will only be able to set
; environment variables that begin with PHP_ (e.g. PHP_FOO=BAR).
;
; Note:  If this directive is empty, PHP will let the user modify ANY
; environment variable!
safe_mode_allowed_env_vars = PHP_

; This directive contains a comma-delimited list of environment variables that
; the end user won't be able to change using putenv().  These variables will be
; protected even if safe_mode_allowed_env_vars is set to allow to change them.
safe_mode_protected_env_vars = LD_LIBRARY_PATH

; open_basedir, if set, limits all file operations to the defined directory
; and below.  This directive makes most sense if used in a per-directory
; or per-virtualhost web server configuration file. This directive is
; *NOT* affected by whether Safe Mode is turned On or Off.
;open_basedir =

; This directive allows you to disable certain functions for security reasons.
; It receives a comma-delimited list of function names. This directive is
; *NOT* affected by whether Safe Mode is turned On or Off.
disable_functions =

; This directive allows you to disable certain classes for security reasons.
; It receives a comma-delimited list of class names. This directive is
; *NOT* affected by whether Safe Mode is turned On or Off.
disable_classes =

; Colors for Syntax Highlighting mode.  Anything that's acceptable in
; <font color="??????"> would work.
;highlight.string  = #DD0000
;highlight.comment = #FF9900
;highlight.keyword = #007700
;highlight.bg      = #FFFFFF
;highlight.default = #0000BB
;highlight.html    = #000000


;
; Misc
```

```
;
; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header).  It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
expose_php = On


;;;;;;;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;;;;;;;

max_execution_time = 30     ; Maximum execution time of each script, in seconds
max_input_time = 60         ; Maximum amount of time each script may spend
parsing request data
memory_limit = 8M      ; Maximum amount of memory a script may consume
(8MB)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Error handling and logging ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; error_reporting is a bit-field.  Or each number up to get desired error
; reporting level
; E_ALL            - All errors and warnings
; E_ERROR          - fatal run-time errors
; E_WARNING        - run-time warnings (non-fatal errors)
; E_PARSE          - compile-time parse errors
; E_NOTICE         - run-time notices (these are warnings which often result
;                  from a bug in your code, but it's possible that it was
;                  intentional (e.g., using an uninitialized variable and
;                  relying on the fact it's automatically initialized to an
;                  empty string)
; E_STRICT                   - run-time notices, enable to have PHP suggest
changes
;                  to your code which will ensure the best interoperability
;                  and forward compatability of your code
; E_CORE_ERROR     - fatal errors that occur during PHP's initial startup
; E_CORE_WARNING   - warnings (non-fatal errors) that occur during PHP's
;                  initial startup
; E_COMPILE_ERROR  - fatal compile-time errors
; E_COMPILE_WARNING - compile-time warnings (non-fatal errors)
; E_USER_ERROR     - user-generated error message
; E_USER_WARNING   - user-generated warning message
; E_USER_NOTICE    - user-generated notice message
```

```
;
; Examples:
;
;   - Show all errors, except for notices and coding standards warnings
;
;error_reporting = E_ALL & ~E_NOTICE & ~E_STRICT
;
;   - Show all errors, except for notices
;
;error_reporting = E_ALL & ~E_NOTICE
;
;   - Show only errors
;
;error_reporting = E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR
;
;   - Show all errors except for notices and coding standards warnings
;
error_reporting  =  E_ALL & ~E_NOTICE & ~E_STRICT

; Print out errors (as a part of the output).  For production web sites,
; you're strongly encouraged to turn this feature off, and use error logging
; instead (see below).  Keeping display_errors enabled on a production web site
; may reveal security information to end users, such as file paths on your Web
; server, your database schema or other information.
display_errors = On

; Even when display_errors is on, errors that occur during PHP's startup
; sequence are not displayed.  It's strongly recommended to keep
; display_startup_errors off, except for when debugging.
display_startup_errors = Off

; Log errors into a log file (server-specific log, stderr, or error_log (below))
; As stated above, you're strongly advised to use error logging in place of
; error displaying on production web sites.
log_errors = Off

; Set maximum length of log_errors. In error_log information about the source is
; added. The default is 1024 and 0 allows to not apply any maximum length at all.
log_errors_max_len = 1024

; Do not log repeated messages. Repeated errors must occur in same file on
same
; line until ignore_repeated_source is set true.
ignore_repeated_errors = Off

; Ignore source of message when ignoring repeated messages. When this setting
```

```
; is On you will not log errors with repeated messages from different files or
; sourcelines.
ignore_repeated_source = Off

; If this parameter is set to Off, then memory leaks will not be shown (on
; stdout or in the log). This has only effect in a debug compile, and if
; error reporting includes E_WARNING in the allowed list
report_memleaks = On

; Store the last error/warning message in $php_errormsg (boolean).
track_errors = Off

; Disable the inclusion of HTML tags in error messages.
; Note: Never use this feature for production boxes.
;html_errors = Off

; If html_errors is set On PHP produces clickable error messages that direct
; to a page describing the error or function causing the error in detail.
; You can download a copy of the PHP manual from http://www.php.net/docs.php
; and change docref_root to the base URL of your local copy including the
; leading '/'. You must also specify the file extension being used including
; the dot.
; Note: Never use this feature for production boxes.
;docref_root = "/phpmanual/"
;docref_ext = .html

; String to output before an error message.
;error_prepend_string = "<font color=ff0000>"

; String to output after an error message.
;error_append_string = "</font>"

; Log errors to specified file.
;error_log = filename

; Log errors to syslog (Event Log on NT, not valid in Windows 95).
;error_log = syslog


;;;;;;;;;;;;;;;;;
; Data Handling ;
;;;;;;;;;;;;;;;;;
;
; Note - track_vars is ALWAYS enabled as of PHP 4.0.3

; The separator used in PHP generated URLs to separate arguments.
```

```
; Default is "&".
;arg_separator.output = "&amp;"

; List of separator(s) used by PHP to parse input URLs into variables.
; Default is "&".
; NOTE: Every character in this directive is considered as separator!
;arg_separator.input = ";&"

; This directive describes the order in which PHP registers GET, POST, Cookie,
; Environment and Built-in variables (G, P, C, E & S respectively, often
; referred to as EGPCS or GPC).  Registration is done from left to right, newer
; values override older values.
variables_order = "EGPCS"

; Whether or not to register the EGPCS variables as global variables.  You may
; want to turn this off if you don't want to clutter your scripts' global scope
; with user data.  This makes most sense when coupled with track_vars - in
which
; case you can access all of the GPC variables through the $HTTP_*_VARS[],
; variables.
;
; You should do your best to write your scripts so that they do not require
; register_globals to be on;  Using form variables as globals can easily lead
; to possible security problems, if the code is not very well thought of.
register_globals = Off

; Whether or not to register the old-style input arrays, HTTP_GET_VARS
; and friends.  If you're not using them, it's recommended to turn them off,
; for performance reasons.
register_long_arrays = On

; This directive tells PHP whether to declare the argv&argc variables (that
; would contain the GET information).  If you don't use these variables, you
; should turn it off for increased performance.
register_argc_argv = On

; Maximum size of POST data that PHP will accept.
post_max_size = 8M

; Magic quotes
;

; Magic quotes for incoming GET/POST/Cookie data.
magic_quotes_gpc = On

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc.
```

```
magic_quotes_runtime = Off

; Use Sybase-style magic quotes (escape ' with '' instead of \').
magic_quotes_sybase = Off

; Automatically add files before or after any PHP document.
auto_prepend_file =
auto_append_file =

; As of 4.0b4, PHP always outputs a character encoding by default in
; the Content-type: header.  To disable sending of the charset, simply
; set it to be empty.
;
; PHP's built-in default is text/html
default_mimetype = "text/html"
;default_charset = "iso-8859-1"

; Always populate the $HTTP_RAW_POST_DATA variable.
;always_populate_raw_post_data = On


;;;;;;;;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;;;;;;;

; UNIX: "/path1:/path2"
;include_path = ".:/php/includes"
;
; Windows: "\path1;\path2"
;include_path = ".;c:\php\includes"

; The root of the PHP pages, used only if nonempty.
; if PHP was not compiled with FORCE_REDIRECT, you SHOULD set doc_root
; if you are running php as a CGI under any web server (other than IIS)
; see documentation for security issues.  The alternate is to use the
; cgi.force_redirect configuration below
doc_root =

; The directory under which PHP opens the script using /~username used only
; if nonempty.
user_dir =

; Directory in which the loadable extensions (modules) reside.
extension_dir = "./"

; Whether or not to enable the dl() function.  The dl() function does NOT work
```

```
; properly in multithreaded servers, such as IIS or Zeus, and is automatically
; disabled on them.
enable_dl = On

; cgi.force_redirect is necessary to provide security running PHP as a CGI under
; most web servers.  Left undefined, PHP turns this on by default.  You can
; turn it off here AT YOUR OWN RISK
; **You CAN safely turn this off for IIS, in fact, you MUST.**
; cgi.force_redirect = 1

; if cgi.nph is enabled it will force cgi to always sent Status: 200 with
; every request.
; cgi.nph = 1

; if cgi.force_redirect is turned on, and you are not running under Apache or
Netscape
; (iPlanet) web servers, you MAY need to set an environment variable name that
PHP
; will look for to know it is OK to continue execution.  Setting this variable MAY
; cause security issues, KNOW WHAT YOU ARE DOING FIRST.
; cgi.redirect_status_env = ;

; FastCGI under IIS (on WINNT based OS) supports the ability to impersonate
; security tokens of the calling client.  This allows IIS to define the
; security context that the request runs under.  mod_fastcgi under Apache
; does not currently support this feature (03/17/2002)
; Set to 1 if running under IIS.  Default is zero.
; fastcgi.impersonate = 1;

; cgi.rfc2616_headers configuration option tells PHP what type of headers to
; use when sending HTTP response code. If it's set 0 PHP sends Status: header
that
; is supported by Apache. When this option is set to 1 PHP will send
; RFC2616 compliant header.
; Default is zero.
;cgi.rfc2616_headers = 0


;;;;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;;;;

; Whether to allow HTTP file uploads.
file_uploads = On

; Temporary directory for HTTP uploaded files (will use system default if not
```

```
; specified).
;upload_tmp_dir =

; Maximum allowed size for uploaded files.
upload_max_filesize = 2M


;;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
allow_url_fopen = On

; Define the anonymous ftp password (your email address)
;from="john@doe.com"

; Define the User-Agent string
; user_agent="PHP"

; Default timeout for socket based streams (seconds)
default_socket_timeout = 60

; If your scripts have to deal with files from Macintosh systems,
; or you are running on a Mac and need to deal with files from
; unix or win32 systems, setting this flag will cause PHP to
; automatically detect the EOL character in those files so that
; fgets() and file() will work regardless of the source of the file.
; auto_detect_line_endings = Off


;;;;;;;;;;;;;;;;;;;;;
; Dynamic Extensions ;
;;;;;;;;;;;;;;;;;;;;;
;
; If you wish to have an extension loaded automatically, use the following
; syntax:
;
;   extension=modulename.extension
;
; For example, on Windows:
;
;   extension=msql.dll
;
; ... or under UNIX:
;
```

```
;   extension=msql.so
;
; Note that it should be the name of the module only; no directory information
; needs to go here.  Specify the location of the extension with the
; extension_dir directive above.


;Windows Extensions
;Note that ODBC support is built in, so no dll is needed for it.
;

;extension=php_bz2.dll
;extension=php_cpdf.dll
;extension=php_curl.dll
;extension=php_dba.dll
;extension=php_dbase.dll
;extension=php_dbx.dll
;extension=php_exif.dll
;extension=php_fdf.dll
;extension=php_filepro.dll
;extension=php_gd2.dll
;extension=php_gettext.dll
;extension=php_iconv.dll
;extension=php_ifx.dll
;extension=php_iisfunc.dll
;extension=php_imap.dll
;extension=php_interbase.dll
;extension=php_java.dll
;extension=php_ldap.dll
;extension=php_mbstring.dll
;extension=php_mcrypt.dll
;extension=php_mhash.dll
;extension=php_mime_magic.dll
;extension=php_ming.dll
;extension=php_mssql.dll
;extension=php_msql.dll
;extension=php_mysql.dll
;extension=php_oci8.dll
;extension=php_openssl.dll
;extension=php_oracle.dll
;extension=php_pdf.dll
;extension=php_pgsql.dll
;extension=php_shmop.dll
;extension=php_snmp.dll
;extension=php_sockets.dll
;extension=php_sybase_ct.dll
```

```
;extension=php_tidy.dll
;extension=php_w32api.dll
;extension=php_xmlrpc.dll
;extension=php_xsl.dll
;extension=php_yaz.dll
;extension=php_zip.dll


;;;;;;;;;;;;;;;;;;;
; Module Settings ;
;;;;;;;;;;;;;;;;;;;

[Syslog]
; Whether or not to define the various syslog variables (e.g. $LOG_PID,
; $LOG_CRON, etc.).  Turning it off is a good idea performance-wise.  In
; runtime, you can define these variables by calling define_syslog_variables().
define_syslog_variables  = Off

[mail function]
; For Win32 only.
SMTP = localhost
smtp_port = 25

; For Win32 only.
;sendmail_from = me@example.com

; For Unix only.  You may supply arguments as well (default: "sendmail -t -i").
;sendmail_path =

; Force the addition of the specified parameters to be passed as extra
parameters
; to the sendmail binary. These parameters will always replace the value of
; the 5th parameter to mail(), even in safe mode.
;mail.force_extra_paramaters =

[SQL]
sql.safe_mode = Off

[ODBC]
;odbc.default_db    =  Not yet implemented
;odbc.default_user  =  Not yet implemented
;odbc.default_pw    =  Not yet implemented

; Allow or prevent persistent links.
odbc.allow_persistent = On
```

```
; Check that a connection is still valid before reuse.
odbc.check_persistent = On

; Maximum number of persistent links.  -1 means no limit.
odbc.max_persistent = -1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.
odbc.max_links = -1

; Handling of LONG fields.  Returns number of bytes to variables.  0 means
; passthru.
odbc.defaultlrl = 4096

; Handling of binary data.  0 means passthru, 1 return as is, 2 convert to char.
; See the documentation on odbc_binmode and odbc_longreadlen for an
explanation
; of uodbc.defaultlrl and uodbc.defaultbinmode
odbc.defaultbinmode = 1

[MySQL]
; Allow or prevent persistent links.
mysql.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
mysql.max_persistent = -1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.
mysql.max_links = -1

; Default port number for mysql_connect().  If unset, mysql_connect() will use
; the $MYSQL_TCP_PORT or the mysql-tcp entry in /etc/services or the
; compile-time value defined MYSQL_PORT (in that order).  Win32 will only look
; at MYSQL_PORT.
mysql.default_port =

; Default socket name for local MySQL connects.  If empty, uses the built-in
; MySQL defaults.
mysql.default_socket =

; Default host for mysql_connect() (doesn't apply in safe mode).
mysql.default_host =

; Default user for mysql_connect() (doesn't apply in safe mode).
mysql.default_user =

; Default password for mysql_connect() (doesn't apply in safe mode).
```

```
; Note that this is generally a *bad* idea to store passwords in this file.
; *Any* user with PHP access can run 'echo
get_cfg_var("mysql.default_password")
; and reveal this password!  And of course, any users with read access to this
; file will be able to reveal the password as well.
mysql.default_password =

; Maximum time (in secondes) for connect timeout. -1 means no limimt
mysql.connect_timeout = 60

; Trace mode. When trace_mode is active (=On), warnings for table/index scans
and
; SQL-Erros will be displayed.
mysql.trace_mode = Off

[MySQLI]

; Maximum number of links.  -1 means no limit.
mysqli.max_links = -1

; Default port number for mysqli_connect().  If unset, mysqli_connect() will use
; the $MYSQL_TCP_PORT or the mysql-tcp entry in /etc/services or the
; compile-time value defined MYSQL_PORT (in that order).  Win32 will only look
; at MYSQL_PORT.
mysqli.default_port = 3306

; Default socket name for local MySQL connects.  If empty, uses the built-in
; MySQL defaults.
mysqli.default_socket =

; Default host for mysql_connect() (doesn't apply in safe mode).
mysqli.default_host =

; Default user for mysql_connect() (doesn't apply in safe mode).
mysqli.default_user =

; Default password for mysqli_connect() (doesn't apply in safe mode).
; Note that this is generally a *bad* idea to store passwords in this file.
; *Any* user with PHP access can run 'echo
get_cfg_var("mysqli.default_password")
; and reveal this password!  And of course, any users with read access to this
; file will be able to reveal the password as well.
mysqli.default_password =

; Allow or prevent reconnect
mysqli.reconnect = Off
```

```
[mSQL]
; Allow or prevent persistent links.
msql.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
msql.max_persistent = -1

; Maximum number of links (persistent+non persistent).  -1 means no limit.
msql.max_links = -1

[PostgresSQL]
; Allow or prevent persistent links.
pgsql.allow_persistent = On

; Detect broken persistent links always with pg_pconnect().
; Auto reset feature requires a little overheads.
pgsql.auto_reset_persistent = Off

; Maximum number of persistent links.  -1 means no limit.
pgsql.max_persistent = -1

; Maximum number of links (persistent+non persistent).  -1 means no limit.
pgsql.max_links = -1

; Ignore PostgreSQL backends Notice message or not.
; Notice message logging require a little overheads.
pgsql.ignore_notice = 0

; Log PostgreSQL backends Noitce message or not.
; Unless pgsql.ignore_notice=0, module cannot log notice message.
pgsql.log_notice = 0

[Sybase]
; Allow or prevent persistent links.
sybase.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
sybase.max_persistent = -1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.
sybase.max_links = -1

;sybase.interface_file = "/usr/sybase/interfaces"

; Minimum error severity to display.
```

```
sybase.min_error_severity = 10

; Minimum message severity to display.
sybase.min_message_severity = 10

; Compatability mode with old versions of PHP 3.0.
; If on, this will cause PHP to automatically assign types to results according
; to their Sybase type, instead of treating them all as strings.  This
; compatability mode will probably not stay around forever, so try applying
; whatever necessary changes to your code, and turn it off.
sybase.compatability_mode = Off

[Sybase-CT]
; Allow or prevent persistent links.
sybct.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
sybct.max_persistent = -1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.
sybct.max_links = -1

; Minimum server message severity to display.
sybct.min_server_severity = 10

; Minimum client message severity to display.
sybct.min_client_severity = 10

[dbx]
; returned column names can be converted for compatibility reasons
; possible values for dbx.colnames_case are
; "unchanged" (default, if not set)
; "lowercase"
; "uppercase"
; the recommended default is either upper- or lowercase, but
; unchanged is currently set for backwards compatibility
dbx.colnames_case = "unchanged"

[bcmath]
; Number of decimal digits for all bcmath functions.
bcmath.scale = 0

[browscap]
;browscap = extra/browscap.ini

[Informix]
```

```
; Default host for ifx_connect() (doesn't apply in safe mode).
ifx.default_host =

; Default user for ifx_connect() (doesn't apply in safe mode).
ifx.default_user =

; Default password for ifx_connect() (doesn't apply in safe mode).
ifx.default_password =

; Allow or prevent persistent links.
ifx.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
ifx.max_persistent = -1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.
ifx.max_links = -1

; If on, select statements return the contents of a text blob instead of its id.
ifx.textasvarchar = 0

; If on, select statements return the contents of a byte blob instead of its id.
ifx.byteasvarchar = 0

; Trailing blanks are stripped from fixed-length char columns.  May help the
; life of Informix SE users.
ifx.charasvarchar = 0

; If on, the contents of text and byte blobs are dumped to a file instead of
; keeping them in memory.
ifx.blobinfile = 0

; NULL's are returned as empty strings, unless this is set to 1.  In that case,
; NULL's are returned as string 'NULL'.
ifx.nullformat = 0

[Session]
; Handler used to store/retrieve data.
session.save_handler = files

; Argument passed to save_handler.  In the case of files, this is the path
; where data files are stored. Note: Windows users have to change this
; variable in order to use PHP's session functions.
;
; As of PHP 4.0.1, you can define the path as:
;
```

```
;     session.save_path = "N;/path"
;
; where N is an integer.  Instead of storing all the session files in
; /path, what this will do is use subdirectories N-levels deep, and
; store the session data in those directories.  This is useful if you
; or your OS have problems with lots of files in one directory, and is
; a more efficient layout for servers that handle lots of sessions.
;
; NOTE 1: PHP will not create this directory structure automatically.
;         You can use the script in the ext/session dir for that purpose.
; NOTE 2: See the section on garbage collection below if you choose to
;         use subdirectories for session storage
;
; The file storage module creates files using mode 600 by default.
; You can change that by using
;
;     session.save_path = "N;MODE;/path"
;
; where MODE is the octal representation of the mode. Note that this
; does not overwrite the process's umask.
;session.save_path = "/tmp"

; Whether to use cookies.
session.use_cookies = 1

; This option enables administrators to make their users invulnerable to
; attacks which involve passing session ids in URLs; defaults to 0.
; session.use_only_cookies = 1

; Name of the session (used as cookie name).
session.name = PHPSESSID

; Initialize session on request startup.
session.auto_start = 0

; Lifetime in seconds of cookie or, if 0, until browser is restarted.
session.cookie_lifetime = 0

; The path for which the cookie is valid.
session.cookie_path = /

; The domain for which the cookie is valid.
session.cookie_domain =

; Handler used to serialize data.  php is the standard serializer of PHP.
session.serialize_handler = php
```

```
; Define the probability that the 'garbage collection' process is started
; on every session initialization.
; The probability is calculated by using gc_probability/gc_divisor,
; e.g. 1/100 means there is a 1% chance that the GC process starts
; on each request.

session.gc_probability = 1
session.gc_divisor     = 100

; After this number of seconds, stored data will be seen as 'garbage' and
; cleaned up by the garbage collection process.
session.gc_maxlifetime = 1440

; NOTE: If you are using the subdirectory option for storing session files
;       (see session.save_path above), then garbage collection does *not*
;       happen automatically.  You will need to do your own garbage
;       collection through a shell script, cron entry, or some other method.
;       For example, the following script would is the equivalent of
;       setting session.gc_maxlifetime to 1440 (1440 seconds = 24 minutes):
;          cd /path/to/sessions; find -cmin +24 | xargs rm

; PHP 4.2 and less have an undocumented feature/bug that allows you to
; to initialize a session variable in the global scope, albeit register_globals
; is disabled.  PHP 4.3 and later will warn you, if this feature is used.
; You can disable the feature and the warning seperately. At this time,
; the warning is only displayed, if bug_compat_42 is enabled.

session.bug_compat_42 = 1
session.bug_compat_warn = 1

; Check HTTP Referer to invalidate externally stored URLs containing ids.
; HTTP_REFERER has to contain this substring for the session to be
; considered as valid.
session.referer_check =

; How many bytes to read from the file.
session.entropy_length = 0

; Specified here to create the session id.
session.entropy_file =

;session.entropy_length = 16

;session.entropy_file = /dev/urandom
```

```
; Set to {nocache,private,public,} to determine HTTP caching aspects
; or leave this empty to avoid sending anti-caching headers.
session.cache_limiter = nocache

; Document expires after n minutes.
session.cache_expire = 180

; trans sid support is disabled by default.
; Use of trans sid may risk your users security.
; Use this option with caution.
; - User may send URL contains active session ID
;   to other person via. email/irc/etc.
; - URL that contains active session ID may be stored
;   in publically accessible computer.
; - User may access your site with the same session ID
;   always using URL stored in browser's history or bookmarks.
session.use_trans_sid = 0

; Select a hash function
; 0: MD5   (128 bits)
; 1: SHA-1 (160 bits)
session.hash_function = 0

; Define how many bits are stored in each character when converting
; the binary hash data to something readable.
;
; 4 bits: 0-9, a-f
; 5 bits: 0-9, a-v
; 6 bits: 0-9, a-z, A-Z, "-", ","
session.hash_bits_per_character = 4

; The URL rewriter will look for URLs in a defined set of HTML tags.
; form/fieldset are special; if you include them here, the rewriter will
; add a hidden <input> field with the info which is otherwise appended
; to URLs.  If you want XHTML conformity, remove the form entry.
; Note that all valid entries require a "=", even if no value follows.
url_rewriter.tags = "a=href,area=href,frame=src,input=src,form=,fieldset="

[MSSQL]
; Allow or prevent persistent links.
mssql.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
mssql.max_persistent = -1

; Maximum number of links (persistent+non persistent).  -1 means no limit.
```

```
mssql.max_links = -1

; Minimum error severity to display.
mssql.min_error_severity = 10

; Minimum message severity to display.
mssql.min_message_severity = 10

; Compatability mode with old versions of PHP 3.0.
mssql.compatability_mode = Off

; Connect timeout
;mssql.connect_timeout = 5

; Query timeout
;mssql.timeout = 60

; Valid range 0 - 2147483647.  Default = 4096.
;mssql.textlimit = 4096

; Valid range 0 - 2147483647.  Default = 4096.
;mssql.textsize = 4096

; Limits the number of records in each batch.  0 = all records in one batch.
;mssql.batchsize = 0

; Specify how datetime and datetim4 columns are returned
; On => Returns data converted to SQL server settings
; Off => Returns values as YYYY-MM-DD hh:mm:ss
;mssql.datetimeconvert = On

; Use NT authentication when connecting to the server
mssql.secure_connection = Off

; Specify max number of processes. Default = 25
;mssql.max_procs = 25

[Assertion]
; Assert(expr); active by default.
;assert.active = On

; Issue a PHP warning for each failed assertion.
;assert.warning = On

; Don't bail out by default.
;assert.bail = Off
```

```
; User-function to be called if an assertion fails.
;assert.callback = 0

; Eval the expression with current error_reporting().  Set to true if you want
; error_reporting(0) around the eval().
;assert.quiet_eval = 0

[Ingres II]
; Allow or prevent persistent links.
ingres.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
ingres.max_persistent = -1

; Maximum number of links, including persistents.  -1 means no limit.
ingres.max_links = -1

; Default database (format: [node_id::]dbname[/srv_class]).
ingres.default_database =

; Default user.
ingres.default_user =

; Default password.
ingres.default_password =

[Verisign Payflow Pro]
; Default Payflow Pro server.
pfpro.defaulthost = "test-payflow.verisign.com"

; Default port to connect to.
pfpro.defaultport = 443

; Default timeout in seconds.
pfpro.defaulttimeout = 30

; Default proxy IP address (if required).
;pfpro.proxyaddress =

; Default proxy port.
;pfpro.proxyport =

; Default proxy logon.
;pfpro.proxylogon =
```

```
; Default proxy password.
;pfpro.proxypassword =

[Sockets]
; Use the system read() function instead of the php_read() wrapper.
sockets.use_system_read = On

[com]
; path to a file containing GUIDs, IIDs or filenames of files with TypeLibs
;com.typelib_file =
; allow Distributed-COM calls
;com.allow_dcom = true
; autoregister constants of a components typlib on com_load()
;com.autoregister_typelib = true
; register constants casesensitive
;com.autoregister_casesensitive = false
; show warnings on duplicate constat registrations
;com.autoregister_verbose = true

[mbstring]
; language for internal character representation.
;mbstring.language = Japanese

; internal/script encoding.
; Some encoding cannot work as internal encoding.
; (e.g. SJIS, BIG5, ISO-2022-*)
;mbstring.internal_encoding = EUC-JP

; http input encoding.
;mbstring.http_input = auto

; http output encoding. mb_output_handler must be
; registered as output buffer to function
;mbstring.http_output = SJIS

; enable automatic encoding translation accoding to
; mbstring.internal_encoding setting. Input chars are
; converted to internal encoding by setting this to On.
; Note: Do _not_ use automatic encoding translation for
;       portable libs/applications.
;mbstring.encoding_translation = Off

; automatic encoding detection order.
; auto means
;mbstring.detect_order = auto
```

```
; substitute_character used when character cannot be converted
; one from another
;mbstring.substitute_character = none;

; overload(replace) single byte functions by mbstring functions.
; mail(), ereg(), etc are overloaded by mb_send_mail(), mb_ereg(),
; etc. Possible values are 0,1,2,4 or combination of them.
; For example, 7 for overload everything.
; 0: No overload
; 1: Overload mail() function
; 2: Overload str*() functions
; 4: Overload ereg*() functions
;mbstring.func_overload = 0

[FrontBase]
;fbsql.allow_persistent = On
;fbsql.autocommit = On
;fbsql.default_database =
;fbsql.default_database_password =
;fbsql.default_host =
;fbsql.default_password =
;fbsql.default_user = "_SYSTEM"
;fbsql.generate_warnings = Off
;fbsql.max_connections = 128
;fbsql.max_links = 128
;fbsql.max_persistent = -1
;fbsql.max_results = 128
;fbsql.batchSize = 1000

[exif]
; Exif UNICODE user comments are handled as UCS-2BE/UCS-2LE and JIS as
JIS.
; With mbstring support this will automatically be converted into the encoding
; given by corresponding encode setting. When empty
mbstring.internal_encoding
; is used. For the decode settings you can distinguish between motorola and
; intel byte order. A decode setting cannot be empty.
;exif.encode_unicode = ISO-8859-15
;exif.decode_unicode_motorola = UCS-2BE
;exif.decode_unicode_intel    = UCS-2LE
;exif.encode_jis =
;exif.decode_jis_motorola = JIS
;exif.decode_jis_intel    = JIS

[Tidy]
; The path to a default tidy configuration file to use when using tidy
```

```
;tidy.default_config = /usr/local/lib/php/default.tcfg

; Should tidy clean and repair output automatically?
; WARNING: Do not use this option if you are generating non-html content
; such as dynamic images
tidy.clean_output = Off

[soap]
; Enables or disables WSDL caching feature.
soap.wsdl_cache_enabled=1
; Sets the directory name where SOAP extension will put cache files.
soap.wsdl_cache_dir="/tmp"
; (time to live) Sets the number of second while cached file will be used
; instead of original one.
soap.wsdl_cache_ttl=86400

; Local Variables:
; tab-width: 4
; End:
```

---

[i] T . Berners-Lee, http://www.faqs.org/rfcs/rfc2616.html and R. Fielding, http://www.faqs.org/rfcs/rfc1945.html

[ii] http://www.faqs.org/rfcs/rfc793.html

[iii] http://www.faqs.org/rfcs/rfc791.html

[iv] T. Berners-Lee, http://www.faqs.org/rfcs/rfc1945.html

[v] T. Berners-Lee, http://www.faqs.org/rfcs/rfc1945.html

[vi] T. Berners-Lee, http://www.faqs.org/rfcs/rfc1945.html

[vii] E. Rescorla, http://www.faqs.org/rfcs/rfc2818.html

[viii] T. Berners-Lee, http://www.faqs.org/rfcs/rfc1945.html

[ix] D. Raggett, http://www.w3c.org/MarkUp/

[x] B. Bos, http://www.w3.org/TR/CSS21/

[xi] http://devedge.netscape.com/central/javascript/

[xii] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vbscripttoc.asp

[xiii] http://www.ecma-international.org/publications/standards/Ecma-262.htm

[xiv] http://devedge.netscape.com/library/manuals/2000/javascript/1.3/reference/cookies.html

[xv] K. Coar, http://cgi-spec.golux.com/

[xvi] D. Brogdon, http://www.onlamp.com/pub/a/php/2001/03/29/php_admin.html

[xvii] Apache Software Foundation, http://httpd.apache.org/docs/misc/security_tips.html

[xviii] Apache Software Foundation, http://httpd.apache.org

[xix] National Center for Supercomputing Applications, HTTPd home page, http://hoohoo.ncsa.uiuc.edu/
[xx] http://msdn.microsoft.com/library/default.asp?url=/nhp/default.asp?contentid=28000522
[xxi] I. Ristic, http://www.webkreator.com/php/configuration/php-configuration.html
[xxii] http://www.php.net
[xxiii] S. Bakken, PHP Manual, http://www.php.net/manual/en/configuration.php
[xxiv] S. Bakken, PHP Manual, http://www.php.net/manual/en/install.windows.php
[xxv] [xxv] I. Ristic, http://www.webkreator.com/php/configuration/php-configuration.html
[xxvi] http://cvs.php.net/co.php/php-src/php.ini-dist
[xxvii] H. Fuecks, http://www.sitepoint.com/article/php-anthology-1-1-php-basics/6
[xxviii] S. Bakken, PHP Manual, http://us2.php.net/manual/en/features.safe-mode.functions.php
[xxix] S. Bakken, PHP Manual, http://us2.php.net/manual/en/features.safe-mode.functions.php
[xxx] http://cvs.php.net/co.php/php-src/php.ini-dist
[xxxi] http://cvs.php.net/co.php/php-src/php.ini-dist
[xxxii] C. Lonvick, http://www.faqs.org/rfcs/rfc3164.html
[xxxiii] S. Clowes, http://www.securereality.com.au/archives/studyinscarlet.txt
[xxxiv] http://www.microsoft.com/WindowsServer2003/iis/default.mspx
[xxxv] http://us2.php.net/session
[xxxvi] Open Web Application Security Project http://www.owasp.org/documentation/topten
[xxxvii] Y. Youman, http://fort-knox.org/thesis.php
[xxxviii] L. Stein, http://www.w3.org/Security/Faq/wwwsf4.html
[xxxix] R. Schwartz, http://www.stonehenge.com/merlyn/
[xl] http://www.ansi.org/
[xli] http://www.ecma-international.org/publications/standards/Ecma-006.htm
[xlii] http://www.unicode.org/
[xliii] D. Wheeler, http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/php.html

Last Updated: September 20th, 2017

Upcoming Training

Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS New York SEC401*	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Mentor Session - SEC401	Minneapolis, MN	Oct 03, 2017 - Nov 14, 2017	Mentor
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS Tysons Corner Fall 2017	McLean, VA	Oct 14, 2017 - Oct 21, 2017	Live Event
SANS Tokyo Autumn 2017	Tokyo, Japan	Oct 16, 2017 - Oct 28, 2017	Live Event
CCB Private SEC401 Oct 17	Brussels, Belgium	Oct 16, 2017 - Oct 21, 2017	
Community SANS Omaha SEC401	Omaha, NE	Oct 23, 2017 - Oct 28, 2017	Community SANS
SANS vLive - SEC401: Security Essentials Bootcamp Style	SEC401 - 201710,	Oct 23, 2017 - Nov 29, 2017	vLive
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
San Diego Fall 2017 - SEC401: Security Essentials Bootcamp Style	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	vLive
SANS Gulf Region 2017	Dubai, United Arab Emirates	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Miami 2017	Miami, FL	Nov 06, 2017 - Nov 11, 2017	Live Event
Community SANS Vancouver SEC401*	Vancouver, BC	Nov 06, 2017 - Nov 11, 2017	Community SANS
Community SANS Colorado Springs SEC401**	Colorado Springs, CO	Nov 06, 2017 - Nov 11, 2017	Community SANS
SANS Paris November 2017	Paris, France	Nov 13, 2017 - Nov 18, 2017	Live Event
SANS Sydney 2017	Sydney, Australia	Nov 13, 2017 - Nov 25, 2017	Live Event
SANS San Francisco Winter 2017	San Francisco, CA	Nov 27, 2017 - Dec 02, 2017	Live Event
SANS London November 2017	London, United Kingdom	Nov 27, 2017 - Dec 02, 2017	Live Event
Community SANS St. Louis SEC401	St Louis, MO	Nov 27, 2017 - Dec 02, 2017	Community SANS
Community SANS Portland SEC401	Portland, OR	Nov 27, 2017 - Dec 02, 2017	Community SANS
SANS Khobar 2017	Khobar, Saudi Arabia	Dec 02, 2017 - Dec 07, 2017	Live Event
SANS Munich December 2017	Munich, Germany	Dec 04, 2017 - Dec 09, 2017	Live Event