Linux: 25 PHP Security Best Practices For Sys Admins

by Vivek Gite on November 23, 2011

PHP is an open-source server-side scripting language and it is a widely used. The Apache web server provides access to files and content via the HTTP OR HTTPS protocol. A misconfigured server-side scripting language can create all sorts of problems. So, PHP should be used with caution. Here are twenty-five **php security best practices for sysadmins** for configuring PHP securely.

Our Sample Setup For PHP Security Tips

- DocumentRoot: /var/www/html
- Default Web server: Apache (you can use Lighttpd or Nginx instead of Apache)
- Default PHP configuration file: /etc/php.ini
- Default PHP extensions config directory: /etc/php.d/
- Our sample php security config file: **/etc/php.d/security.ini** (you need to create this file using a text editor)
- Operating systems: RHEL / CentOS / Fedora Linux (the instructions should work with *any other Linux distributions* such as Debian / Ubuntu or other *Unix* like operating systems such as OpenBSD/FreeBSD/HP-UX).
- Default php server TCP/UDP ports: none

Most of the actions listed in this post are written with the assumption that they will be executed by the root user running the bash or any other modern shell:

```
$ php -v
```

Sample outputs:

```
PHP 5.3.3 (cli) (built: Oct 24 2011 08:35:41)
Copyright (c) 1997-2010 The PHP Group
Zend Engine v2.3.0, Copyright (c) 1998-2010 Zend Technologies
```

For demonstration purpose I'm going to use the following operating system:

```
$ cat /etc/redhat-release
```

Sample outputs:

```
Red Hat Enterprise Linux Server release 6.1 (Santiago)
```

#1: Know Your Enemy

PHP based apps can face the different types of attacks. I have noticed the different types of attacks:

1. XSS - Cross-site scripting is a vulnerability in php web applications, which attackers may exploit to steal users' information. You can configure Apache and write more secure PHP scripts (validating all user input) to avoid xss attacks.
2. SQL injection - It is a vulnerability in the database layer of an php application. When user input is incorrectly filtered any SQL statements can be executed by the application. You can configure Apache and write secure code (validating and escaping all user input) to avoid SQL injection attacks. A common practice in PHP is to escape parameters using the function called **mysql_real_escape_string()** before sending the SQL query.
   Spoofing
3. File uploads - It allows your visitor to place files (upload files) on your server. This can result into various security problems such as delete your files, delete database, get user details and much more. You can disable file uploads using php or write secure code (like validating user input and only allow image file type such as png or gif).
4. Including local and remote files - An attacker can open files from remote server and execute any PHP code. This allows them to upload file, delete file and install backdoors. You can configure php to disable remote file execution.
5. eval() - Evaluate a string as PHP code. This is often used by an attacker to hide their code and tools on the server itself. You can configure php to disable eval().
6. Sea-surf Attack (Cross-site request forgery - CSRF) - This attack forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. A successful CSRF exploit can compromise end user data and operation in case of normal user. If the targeted end user is the administrator account, this can compromise the entire web application.

#2: Find Built-in PHP Modules

To see the set of compiled-in PHP modules type the following command:

```
# php -m
```

Sample outputs:

```
[PHP Modules]
apc
bcmath
bz2
calendar
Core
```

```
ctype
curl
date
dom
ereg
exif
fileinfo
filter
ftp
gd
gettext
gmp
hash
iconv
imap
json
libxml
mbstring
memcache
mysql
mysqli
openssl
pcntl
pcre
PDO
pdo_mysql
pdo_sqlite
Phar
readline
Reflection
session
shmop
SimpleXML
sockets
SPL
sqlite3
standard
suhosin
tokenizer
wddx
xml
xmlreader
xmlrpc
xmlwriter
xsl
zip
zlib
[Zend Modules]
Suhosin
```

I recommends that you use PHP with a reduced modules for performance and security. For example, you can disable sqlite3 module by deleting (removing) configuration file , OR renaming (moving) a file called /etc/php.d/sqlite3.ini as follows:

```
# rm /etc/php.d/sqlite3.ini
```

OR

```
# mv /etc/php.d/sqlite3.ini /etc/php.d/sqlite3.disable
```

Other compiled-in modules can only be removed by reinstallating PHP with a reduced configuration. You can download php source code from php.net and compile it as follows with GD, fastcgi, and MySQL support:

```
./configure --with-libdir=lib64 --with-gd --with-mysql --prefix=/usr --exec-prefix=/usr \
--bindir=/usr/bin --sbindir=/usr/sbin --sysconfdir=/etc --datadir=/usr/share \
--includedir=/usr/include --libexecdir=/usr/libexec --localstatedir=/var \
--sharedstatedir=/usr/com --mandir=/usr/share/man --infodir=/usr/share/info \
--cache-file=../config.cache --with-config-file-path=/etc \
--with-config-file-scan-dir=/etc/php.d  --enable-fastcgi \
--enable-force-cgi-redirect
```

See how to compile and reinstall php on Unix like operating system for more information.

#3: Restrict PHP Information Leakage

To restrict PHP information leakage disable expose_php. Edit /etc/php.d/secutity.ini and set the following directive:

```
expose_php=Off
```

When enabled, expose_php reports to the world that PHP is installed on the server, which includes the PHP version within the HTTP header (e.g., X-Powered-By: PHP/5.3.3). The PHP logo guids (see example) are also exposed, thus appending them to the URL of a PHP enabled site will display the appropriate logo. When expose_php enabled you can see php version using the following command:

```
$ curl -I http://www.cyberciti.biz/index.php
```

Sample outputs:

```
HTTP/1.1 200 OK
X-Powered-By: PHP/5.3.3
Content-type: text/html; charset=UTF-8
Vary: Accept-Encoding, Cookie
X-Vary-Options: Accept-Encoding;list-contains=gzip,Cookie;string-contains=wikiToken;string-contains=wikiLoggedOut;string-contains=wiki_session
Last-Modified: Thu, 03 Nov 2011 22:32:55 GMT
...
```

I also recommend that you setup the ServerTokens and ServerSignature directives in httpd.conf to hide Apache version and other information.

#4: Minimize Loadable PHP Modules (Dynamic Extensions)

PHP supports "Dynamic Extensions". By default, RHEL loads all the extension modules found in /etc/php.d/ directory. To enable or disable a particular

module, just find the configuration file in /etc/php.d/ directory and comment the module name. You can also rename or delete module configuration file. For best PHP performance and security, you should only enable the extensions your webapps requires. For example, to disable gd extension, type the following commands:

```
# cd /etc/php.d/
# mv gd.{ini,disable}
# /sbin/service httpd restart
```

To enable php module called gd, enter:

```
# mv gd.{disable,ini}
# /sbin/service httpd restart
```

#5: Log All PHP Errors

Do not expose PHP error messages to all site visitors. Edit /etc/php.d/security.ini and set the following directive:

```
display_errors=Off
```

Make sure you log all php errors to a log file:

```
log_errors=On
error_log=/var/log/httpd/php_scripts_error.log
```

#6: Disallow Uploading Files

Edit /etc/php.d/security.ini and set the following directive to disable file uploads for security reasons:

```
file_uploads=Off
```

If users of your application need to upload files, turn this feature on by setting upload_max_filesize limits the maximum size of files that PHP will accept through uploads:

```
file_uploads=On
# user can only upload upto 1MB via php
upload_max_filesize=1M
```

#7: Turn Off Remote Code Execution

If enabled, allow_url_fopen allows PHP's file functions -- such as file_get_contents() and the include and require statements -- can retrieve data from remote locations, like an FTP or web site.

The allow_url_fopen option allows PHP's file functions - such as file_get_contents() and the include and require statements - can retrieve data from remote locations using ftp or http protocols. Programmers frequently forget this and don't do proper input filtering when passing user-provided data to these functions, opening them up to code injection vulnerabilities. A large number of code injection vulnerabilities reported in PHP-based web applications are caused by the combination of enabling allow_url_fopen and bad input filtering. Edit /etc/php.d/security.ini and set the following directive:

```
allow_url_fopen=Off
```

I also recommend to disable allow_url_include for security reasons:

```
allow_url_include=Off
```

#8: Enable SQL Safe Mode

Edit /etc/php.d/security.ini and set the following directive:

```
sql.safe_mode=On
```

If turned On, mysql_connect() and mysql_pconnect() ignore any arguments passed to them. Please note that you may have to make some changes to your code. Third party and open source application such as WordPress, and others may not work at all when sql.safe_mode enabled. I also recommend that you turn off magic_quotes_gpc for all php 5.3.x installations as the filtering by it is ineffective and not very robust. mysql_escape_string() and custom filtering functions serve a better purpose (hat tip to Eric Hansen):

```
magic_quotes_gpc=Off
```

#9: Control POST Size

The HTTP POST request method is used when the client (browser or user) needs to send data to the Apache web server as part of the request, such as when uploading a file or submitting a completed form. Attackers may attempt to send oversized POST requests to eat your system resources. You can limit the maximum size POST request that PHP will process. Edit /etc/php.d/security.ini and set the following directive:

```
; Set a realistic value here
post_max_size=1K
```

The 1K sets max size of post data allowed by php apps. This setting also affects file upload. To upload large files, this value must be larger than upload_max_filesize. I also suggest that you limit available methods using Apache web server. Edit, httpd.conf and set the following directive for DocumentRoot /var/www/html:

```
<Directory /var/www/html>
    <LimitExcept GET POST>
        Order allow,deny
    </LimitExcept>
## Add rest of the config goes here... ##
</Directory>
```

#10: Resource Control (DoS Control)

You can set maximum execution time of each php script, in seconds. Another recommend option is to set maximum amount of time each script may spend parsing request data, and maximum amount of memory a script may consume. Edit /etc/php.d/security.ini and set the following directives:

```
# set in seconds
max_execution_time =  30
max_input_time = 30
memory_limit = 40M
```

#11: Install Suhosin Advanced Protection System for PHP

From the project page:

> Suhosin is an advanced protection system for PHP installations. It was designed to protect servers and users from known and unknown flaws in PHP applications and the PHP core. Suhosin comes in two independent parts, that can be used separately or in combination. The first part is a small patch against the PHP core, that implements a few low-level protections against bufferoverflows or format string vulnerabilities and the second part is a powerful PHP extension that implements all the other protections.

See how to install and configure suhosin under Linux operating systems.

#12 Disabling Dangerous PHP Functions

PHP has a lot of functions which can be used to crack your server if not used properly. You can set list of functions in /etc/php.d/security.ini using disable_functions directive:

```
disable_functions =exec,passthru,shell_exec,system,proc_open,popen,curl_exec,curl_multi_exec,parse_ini_file,show_source
```

#13 PHP Fastcgi / CGI - cgi.force_redirect Directive

PHP work with FastCGI. Fascgi reduces the memory footprint of your web server, but still gives you the speed and power of the entire PHP language. You can configure Apache2+PHP+FastCGI or cgi as described here. The configuration directive cgi.force_redirect prevents anyone from calling PHP directly with a URL like http://www.cyberciti.biz/cgi-bin/php/hackerdir/backdoor.php. Turn on cgi.force_redirect for security reasons. Edit /etc/php.d/security.ini and set the following directive:

```
; Enable cgi.force_redirect for security reasons in a typical *Apache+PHP-CGI/FastCGI* setup
cgi.force_redirect=On
```

#14 PHP User and Group ID

mod_fastcgi is a cgi-module for Apache web server. It can connect to an external FASTCGI server. You need to make sure php run as non-root user. If PHP executes as a root or UID under 100, it may access and/or manipulate system files. You must execute PHP CGIs as a non-privileged user using Apache's suEXEC or mod_suPHP. The suEXEC feature provides Apache users the ability to run CGI programs under user IDs different from the user ID of the calling web server. In this example, my php-cgi is running as phpcgi user and apache is running as apache user:

```
# ps aux | grep php-cgi
```

Sample outputs:

```
phpcgi      6012  0.0  0.4 225036 60140 ?        S    Nov22   0:12 /usr/bin/php-cgi
phpcgi      6054  0.0  0.5 229928 62820 ?        S    Nov22   0:11 /usr/bin/php-cgi
phpcgi      6055  0.1  0.4 224944 53260 ?        S    Nov22   0:18 /usr/bin/php-cgi
phpcgi      6085  0.0  0.4 224680 56948 ?        S    Nov22   0:11 /usr/bin/php-cgi
phpcgi      6103  0.0  0.4 224564 57956 ?        S    Nov22   0:11 /usr/bin/php-cgi
phpcgi      6815  0.4  0.5 228556 61220 ?        S    00:52   0:19 /usr/bin/php-cgi
phpcgi      6821  0.3  0.5 228008 61252 ?        S    00:55   0:12 /usr/bin/php-cgi
phpcgi      6823  0.3  0.4 225536 58536 ?        S    00:57   0:13 /usr/bin/php-cgi
```

You can use tool such as spawn-fcgi to spawn remote and local FastCGI processes as phpcgi user (first, add phpcgi user to the system):

```
# spawn-fcgi -a 127.0.0.1 -p 9000 -u phpcgi -g phpcgi -f /usr/bin/php-cgi
```

Now, you can configure Apache, Lighttpd, and Nginx web server to use external php FastCGI running on port 9000 at 127.0.0.1 IP address.

#15 Limit PHP Access To File System

The open_basedir directive set the directories from which PHP is allowed to access files using functions like fopen(), and others. If a file is outside of the paths defined by open_basdir, PHP will refuse to open it. You cannot use a symbolic link as a workaround. For example only allow access to /var/www/html directory and not to /var/www, or /tmp or /etc directories:

```
; Limits the PHP process from accessing files outside
; of specifically designated directories such as /var/www/html/
open_basedir="/var/www/html/"
; ----------------------------------
; Multiple dirs example
; open_basedir="/home/httpd/vhost/cyberciti.biz/html/:/home/httpd/vhost/nixcraft.com/html/:/home/httpd/vhost/theos.in/html/"
; ----------------------------------
```

#16 Session Path

Session support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and

increase the appeal of your web site. This path is defined in /etc/php.ini file and all data related to a particular session will be stored in a file in the directory specified by the session.save_path option. The default is as follows under RHEL/CentOS/Fedora Linux:

```
session.save_path="/var/lib/php/session"
; Set the temporary directory used for storing files when doing file upload
upload_tmp_dir="/var/lib/php/session"
```

Make sure path is **outside /var/www/html** and **not readable or writeable** by any other system users:

```
# ls -Z /var/lib/php/
```

Sample outputs:

```
drwxrwx---. root apache system_u:object_r:httpd_var_run_t:s0 session
```

Note: The -Z option to the ls command display SELinux security context such as file mode, user, group, security context and file name.

#17 Keep PHP, Software, And OS Up to Date

Applying security patches is an important part of maintaining Linux, Apache, PHP, and MySQL server. All php security update should be reviewed and applied as soon as possible using any one of the following tool (if you're installing PHP via a package manager):

```
# yum update
```

OR

```
# apt-get update && apt-get upgrade
```

You can configure Red hat / CentOS / Fedora Linux to send yum package update notification via email. Another option is to apply all security updates via a cron job. Under Debian / Ubuntu Linux you can use apticron to send security notifications.

Note: Check php.net for the most recent release for source code installations.

#18: Restrict File and Directory Access

Make sure you run Apache as a non-root user such as Apache or www. All files and directory should be owned by non-root user (or apache user) under /var/www/html:

```
# chown -R apache:apache /var/www/html/
```

/var/www/html/ is a subdirectory and DocumentRoot which is modifiable by other users since root never executes any files out of there, and shouldn't be creating files in there.

Make sure file permissions are set to 0444 (read-only) under /var/www/html/:

```
# chmod -R 0444 /var/www/html/
```

Make sure all directories permissions are set to 0445 under /var/www/html/:

```
# find /var/www/html/ -type d -print0 | xargs -0 -I {} chmod 0445 {}
```

A Note About Setting Up Correct File Permissions

The chown and chmod command make sures that under no circumstances DocumentRoot or files contained in DocumentRoot are writable by the Web server user apache. Please note that you need to set permissions that makes the most sense for the development model of your website, so feel free to adjust the chown and chmod command as per your requirements. In this example, the Apache server run as apache user. This is configured with the *User* and *Group* directives in your httpd.conf file. The apache user needs to have read access to everything under DocumentRoot but should not have write access to anything.

Make sure httpd.conf has the following directives for restrictive configuration:

```
<Directory / >
    Options None
    AllowOverride None
    Order allow,deny
</Directory>
```

You should only grant write access when required. Some web applications such as wordpress and others may need a caching directory. You can grant a write access to caching directory using the following commands:

```
# chmod a+w /var/www/html/blog/wp-content/cache
### block access to all ###
# echo 'deny from all' > /var/www/html/blog/wp-content/cache/.htaccess
```

#19: Write Protect Apache, PHP, and, MySQL Configuration Files

Use the chattr command to write protect configuration files:

```
# chattr +i /etc/php.ini
# chattr +i /etc/php.d/*
# chattr +i /etc/my.ini
# chattr +i /etc/httpd/conf/httpd.conf
# chattr +i /etc/
```

The chattr command can write protect your php file or files in /var/www/html directory too:

```
# chattr +i /var/www/html/file1.php
# chattr +i /var/www/html/
```

#20: Use Linux Security Extensions (such as SELinux)

Linux comes with various security patches which can be used to guard against misconfigured or compromised server programs. If possible use SELinux and other Linux security extensions to enforce limitations on network and other programs. For example, SELinux provides a variety of security policies for Linux kernel and Apache web server. To list all Apache SELinux protection variables, enter:

```
# getsebool -a | grep httpd
```

Sample outputs:

```
allow_httpd_anon_write --> off
allow_httpd_mod_auth_ntlm_winbind --> off
allow_httpd_mod_auth_pam --> off
allow_httpd_sys_script_anon_write --> off
httpd_builtin_scripting --> on
httpd_can_check_spam --> off
httpd_can_network_connect --> off
httpd_can_network_connect_cobbler --> off
httpd_can_network_connect_db --> off
httpd_can_network_memcache --> off
httpd_can_network_relay --> off
httpd_can_sendmail --> off
httpd_dbus_avahi --> on
httpd_enable_cgi --> on
httpd_enable_ftp_server --> off
httpd_enable_homedirs --> off
httpd_execmem --> off
httpd_read_user_content --> off
httpd_setrlimit --> off
httpd_ssi_exec --> off
httpd_tmp_exec --> off
httpd_tty_comm --> on
httpd_unified --> on
httpd_use_cifs --> off
httpd_use_gpg --> off
httpd_use_nfs --> off
```

To disable Apache cgi support, enter:

```
# setsebool -P httpd_enable_cgi off
```

See Red Hat SELinux guide for more information.

#21 Install Mod_security

ModSecurity is an open source intrusion detection and prevention engine for web applications. You can easily install mod_security under Linux and protect apache and php based apps from xss and various other attacks:

```
## A few Examples ##
# Do not allow to open files in /etc/
SecFilter /etc/

# Stop SQL injection
SecFilter "delete[[:space:]]+from"
SecFilter "select.+from"
```

#22 Run Apache / PHP In a Chroot Jail If Possible

Putting PHP and/or Apache in a chroot jail minimizes the damage done by a potential break-in by isolating the web server to a small section of the filesystem. You can use traditional chroot kind of setup with Apache. However, I recommend FreeBSD jails, XEN virtulization, KVM virtulization, or OpenVZ virtualization which uses the concept of containers.

#23 Use Firewall To Restrict Outgoing Connections

The attacker will download file locally on your web-server using tools such as wget. Use iptables to block outgoing connections from apache user. The ipt_owner module attempts to match various characteristics of the packet creator, for locally generated packets. It is only valid in the OUTPUT chain. In this example, allow vivek user to connect outside using port 80 (useful for RHN or centos repo access):

```
/sbin/iptables -A OUTPUT -o eth0 -m owner --uid-owner vivek -p tcp --dport 80 -m state --state NEW,ESTABLISHED  -j ACCEPT
```

Here is another example that blocks all outgoing connections from apache user except to our own smtp server, and spam validation API service:

```
# ....
/sbin/iptables --new-chain apache_user
/sbin/iptables --append OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
/sbin/iptables --append OUTPUT -m owner --uid-owner apache -j apache_user
# allow apache user to connec to our smtp server
/sbin/iptables --append apache_user -p tcp --syn -d 192.168.1.100 --dport 25 -j RETURN
# Allow apache user to connec to api server for spam validation
/sbin/iptables --append apache_user -p tcp --syn -d  66.135.58.62 --dport 80 -j RETURN
/sbin/iptables --append apache_user -p tcp --syn -d  66.135.58.61 --dport 80 -j RETURN
/sbin/iptables --append apache_user -p tcp --syn -d  72.233.69.89 --dport 80 -j RETURN
/sbin/iptables --append apache_user -p tcp --syn -d  72.233.69.88 --dport 80 -j RETURN
#########################
## Add more rules here ##
#########################
# No editing below
# Drop everything for apache outgoing connection
/sbin/iptables --append apache_user -j REJECT
```

#24 Watch Your Logs & Auditing

Check the apache log file:

```
# tail -f /var/log/httpd/error_log
```

```
# grep 'login.php' /var/log/httpd/error_log
# egrep -i "denied|error|warn" /var/log/httpd/error_log
```

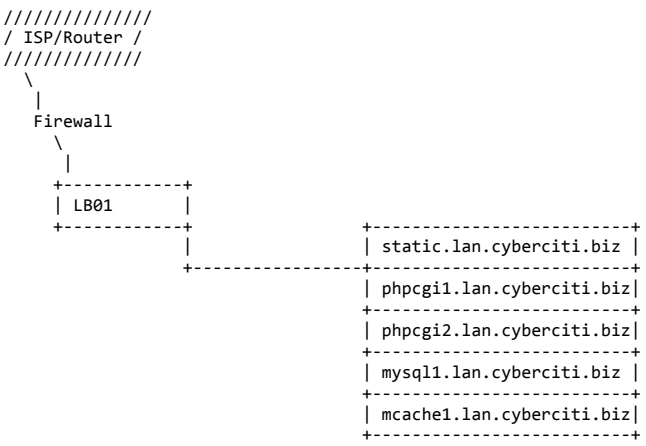
Check the php log file:

```
# tail -f /var/log/httpd/php_scripts_error.log
# grep "...etc/passwd" /var/log/httpd/php_scripts_error.log
```

Log files will give you some understanding of what attacks is thrown against the server and allow you to check if the necessary level of security is present or not. The auditd service is provided for system auditing. Turn it on to audit SELinux events, authetication events, file modifications, account modification and so on. I also recommend using standard "Linux System Monitoring Tools" for monitoring your web-server.

#25 Run Service Per System or VM Instance

For large installations it is recommended that you run, database, static, and dynamic content from different servers.



(Fig.01: Running Services On Separate Servers)

Run different network services on separate servers or VM instances. This limits the number of other services that can be compromised. For example, if an attacker able to successfully exploit a software such as Apache flow, he / she will get an access to entire server including other services running on the same server (such as MySQL, e-mail server and so on). But, in the above example content are served as follows:

1. **static.lan.cyberciti.biz** - Use lighttpd or nginx server for static assets such as js/css/images.
2. **phpcgi1.lan.cyberciti.biz** and **phpcgi2.lan.cyberciti.biz** - Apache web-server with php used for generating dynamic content.
3. **mysql1.lan.cyberciti.biz** - MySQL database server.
4. **mcache1.lan.cyberciti.biz** - Memcached server is very fast caching system for MySQL. It uses libevent or epoll (Linux runtime) to scale to any number of open connections and uses non-blocking network I/O.
5. **LB01** - A nginx web and reverse proxy server in front of Apache Web servers. All connections coming from the Internet addressed to one of the Web servers are routed through the nginx proxy server, which may either deal with the request itself or pass the request wholly or partially to the main web servers. LB01 provides simple load-balancing.

#26 Additional Tools

From the project page:

> PHPIDS (PHP-Intrusion Detection System) is a simple to use, well structured, fast and state-of-the-art security layer for your PHP based web application. The IDS neither strips, sanitizes nor filters any malicious input, it simply recognizes when an attacker tries to break your site and reacts in exactly the way you want it to.

You can use PHPIDS to detect malicious users, and log any attacks detected for later review. Please note that I've personally not used this tool.

From the project page:

> PhpSecInfo provides an equivalent to the phpinfo() function that reports security information about the PHP environment, and offers suggestions for improvement. It is not a replacement for secure development techniques, and does not do any kind of code or app auditing, but can be a useful tool in a multilayered security approach.

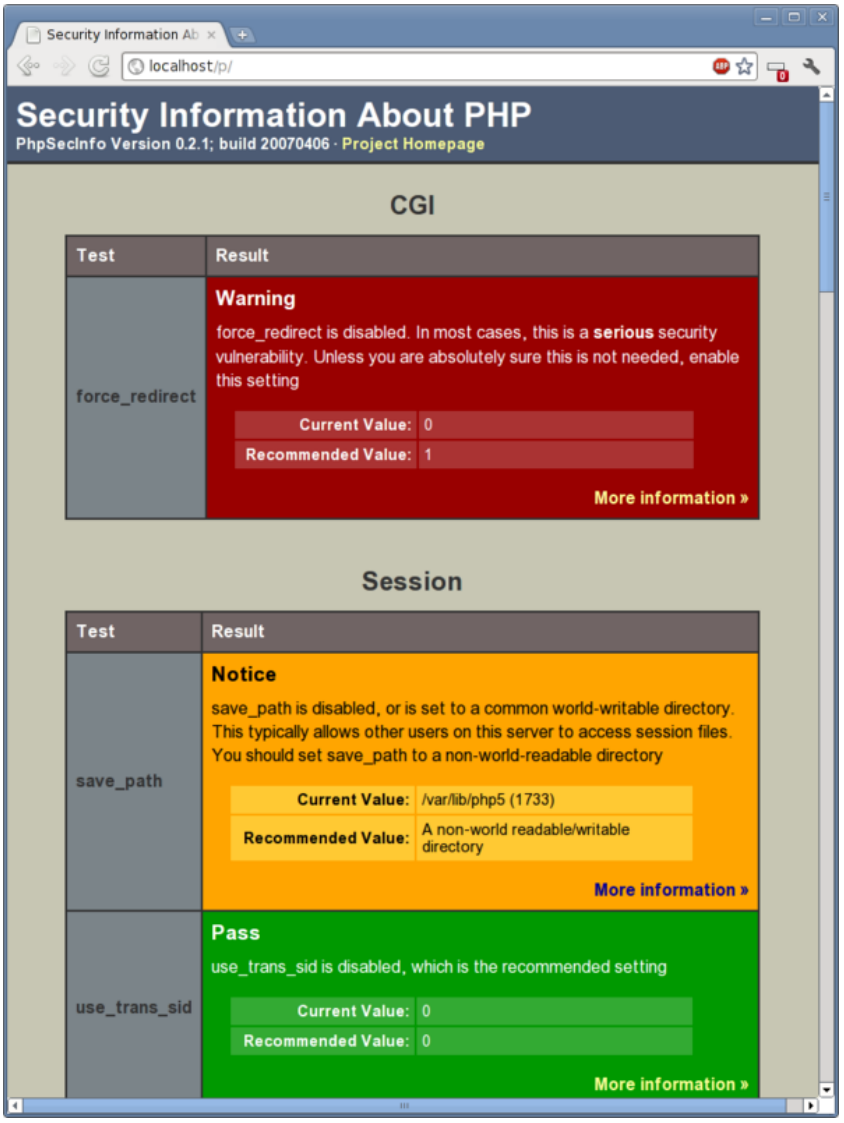


Fig.02: Security Information About PHP Application

See Linux security hardening tips which can reduce available vectors of attack on the system.

A Note About PHP Backdoors

You may come across php scripts or so called common backdoors such as c99, c99madshell, r57 and so on. A backdoor php script is nothing but a hidden script for bypassing all authentication and access your server on demand. It is installed by an attackers to access your server while attempting to remain undetected. Typically a PHP (or any other CGI script) script by mistake allows inclusion of code exploiting vulnerabilities in the web browser. An attacker can use such exploiting vulnerabilities to upload backdoor shells which can give him or her a number of capabilities such as:

- Download files
- Upload files
- Install rootkits
- Set a spam mail servers / relay server
- Set a proxy server to hide tracks
- Take control of server
- Take control of database server
- Steal all information
- Delete all information and database
- Open TCP / UDP ports and much more

Tip: How Do I Search PHP Backdoors?

Use Unix / Linux grep command to search c99 or r57 shell:

```
# grep -iR 'c99' /var/www/html/
# grep -iR 'r57' /var/www/html/
# find /var/www/html/ -name \*.php -type f -print0 | xargs -0 grep c99
# grep -RPn "(passthru|shell_exec|system|base64_decode|fopen|fclose|eval)" /var/www/html/
```

Conclusion

Your PHP based server is now properly harden and ready to show dynamic webpages. However, vulnerabilities are caused mostly by **not following best practice programming rules**. You should be consulted further resources for your web applications security needs especially php programming which is beyond the scope of sys admin work.

References:

1. PHP security - from the official php project.
2. PHP security guide - from the PHP security consortium project.
3. Apache suseexec - documentation from the Apache project.
4. Apache 2.2 - security tips from the Apache project.
5. The Open Web Application Security Project - Common types of application security attacks.

Recommended readings:

1. PHP Security Guide: This guide aims to familiarise you with some of the basic concepts of online security and teach you how to **write more secure PHP scripts**. It's aimed squarely at beginners, but I hope that it still has something to offer more advanced users.
2. Essential PHP Security (kindle edition): A book about web application security written specifically for PHP developers. It covers 30 of the most common and dangerous exploits as well as simple and effective safeguards that protect your PHP applications.
3. SQL Injection Attacks and Defense This book covers sql injection and web-related attacks. It explains SQL injection. How to find, confirm, and automate SQL injection discovery. It has tips and tricks for finding SQL injection within the code. You can create exploits using SQL injection and design to avoid the dangers of these attacks.

Please add your favorite php security tool or tip in the comments.

Updated for accuracy!

Tweet itFacebook itGoogle+ itPDF itFound an error/typo on this page?

Comments on this entry are closed.

- Peter Molnar November 23, 2011, 6:22 am

  You forget one of the most powerful tips: open_basedir. In this case, using /var/www is not the best solution, it would be better as:
  /var/www/website1
  and
  /var/www/website1/www
  /var/www/website1/tmp

  Into apache config:
  php_admin_value open_basedir /var/www/website1
  php_admin_value upload_tmp_dir /var/www/website1/tmp

  So no PHP execution outside the /var/www/website1 directory.

  Link
  - nixCraft November 23, 2011, 10:14 am

    Heh, you read the post before it was finished. It was my fault. I accidentally pressed the Publish button. I appreciate your feedback.

    Link
    - Peter Molnar November 25, 2011, 6:53 am

      I see, it has become “25” instead of “20” that was in my RSS title :)

      Link
- KJBweb November 23, 2011, 10:31 am

  Awesome, I too caught a bit of the post before it was finished but this is a very useful post; posted right whilst I was in the middle of developing an application too, so doubly useful.

  Thanks!

  Link
- TryMe November 23, 2011, 10:55 am

  All php backdoor shell are large in size. Use the following to find it

  ```
  find / -name "*.php" -type f -size +10000k -exec ls -lh {} \; | awk '{ print $9 ": " $5 }'
  find /var/www -name "*.php" -type f -size +10000k -exec ls -lh {} \; | awk '{ print $9 ": " $5 }'
  ```

  Link
  - Yunus November 26, 2011, 6:24 am

    I use http://www.rfxn.com/projects/linux-malware-detect/ which is very useful for detecting PHP backdoors

    Link
- mauri November 23, 2011, 12:00 pm

  What about the use of suPHP ?

  Link
  - Gopihere November 23, 2011, 2:22 pm

    Ya. suPHP is also very useful and helpful for securing PHP websites.

Link

- deady November 23, 2011, 2:34 pm

  apache-itk ++

  Link
- Fred November 23, 2011, 2:36 pm

  This is an outstanding post. One of the best resource I have ever seen regarding PHP security.
  Keep up the good work.!

  Link
- Chaudhary November 23, 2011, 3:41 pm

  awesome, thank you for sharing.

  Link
- Fredrik November 23, 2011, 4:58 pm

  I agree, good post, keep em coming!

  Link
- Firas November 23, 2011, 10:49 pm

  Vivek, what the great VPSs control panel (Secure) works under FreeBSD you prefer?

  Thanks for your post, great job.

  ^F.B

  Link
  - nixCraft November 24, 2011, 7:02 am

    I do not use any control panel under FreeBSD or CentOS/RHEL based systems. Appreciate your post.

    Link
- Umid November 24, 2011, 5:11 am

  I like post very much!!! Thank you!

  Link
- slapper November 24, 2011, 8:30 am

  As usual excellent job !!!

  Link
- Robert Gilaard November 24, 2011, 11:37 am

  Very well written and informative post.

  Do you know what the effect will be on PostgreSQL if you enable SQL safe mode in PHP with the directive:

  sql.safe_mode=On

  Link
  - Tru November 24, 2011, 11:46 am

    This only affects mysql_connect() which is MySQL specific function.

    Link
- Tru November 24, 2011, 11:46 am

  How do you set and use sql.safe_mode? You need to set mysql db setting in httpd.conf:

  ```
  php_admin_value mysql.default_host "192.168.1.5"
  php_admin_value mysql.default_user "DB_USER_LOGIN"
  php_admin_value mysql.default_password "DB_USER_PASSWORD"
  ```

  In php.ini or security.ini:

  ```
  sql.safe_mode=On
  ```

  You app is not aware of the database settings, it consequently cannot disclose them through a bug or a backdoor, unless code injection is possible. In fact, you can enforce that only an ini-based authentication procedure is used by enabling SQL safe mode in PHP via the sql.safe_mode directive. PHP then rejects any database connection attempts that use anything other than ini values for specifying authentication data.

  Source

  Link
- Andres Mujica November 24, 2011, 7:43 pm

  Excellent post, really really good

  thanks for sharing

  Link
- Steve A November 25, 2011, 2:54 pm

Use PDO.

With proper bind variables, SQL injection becomes far less of a problem.

Link

- mario November 26, 2011, 2:07 pm

  Wow, this list has some exceptionally clueless points. There are some good recommendations at the end, but I was expecting magic_quotes halfway in between.

  For example: eval() is not a security issue per se. It's just another name for include(). Randomly disabling modules is as unproductive as disabling file uploads. Mysql_real_escape_string is no longer state of the art; which makes it a bad advise (much less the mysql_escape_string as mentioned later).

  The list in disabling "dangerous functions" is also quite retarded. Not everything with an "exec" in the name does actually call system commands. Disabling "curl_exec" for example will be a pain in the butt if you also disabled "allow_url_fopen". The author knew about "allow_url_include" but bemusingly got the purpose confused here. — Anyway, that's exactly the kind of cursorily security recommendations that makes unacquainted shared hosters go overbroad with limited reasoning.

  Link

  - Jonathan Cremin November 27, 2011, 5:04 pm

    eval() is not another name for include(). eval() is often used by trojan shells, and rarely used wisely or legitimately.

    Link

- bish November 26, 2011, 7:08 pm

  On RHEL/CentOS/Fedora, *never* rebuild an app by hand like php. The version you will end up will be un-tuned, unsupported and very different in features from what the distro offers. It will also within a week need to be built again, tested against the OS and updated! You don't have the time.

  Having said that, don't delete ini files within the php.d ini pool. Don't rename them. Open them up and comment out the parts you like (yes, even if it's everything) and save them back. The reason why has to do with RPM update behaviour when files are missing vs when config files are changed but exist on the system.

  In #23, be careful that you don't take this trick too far. It works because it targets what the apache user can't do. Users who've gained root of course don't have any problem opening their own firewall holes, but you may not have thought as much in the afterglow of reading such a great suggestion.

  It's a nice post. I can see a few things I can definitely use, myself, at home and at work.

  Link

  - Jonathan Cremin November 27, 2011, 5:56 pm

    A year and a half of security fixes since 5.3.3, and you think building it by hand is something you should never do? The *first* thing a responsible sysadmin should do is run a current version of PHP.

    Link

    - bish March 25, 2012, 2:05 am

      I ran into this exact problem not so long ago. Apparently, some people believe that RH and others just compile a 5.3.3. and just leave it. How naive!

      The customer in question had to — HAD TO — have a 'new' PHP 5.3.11, as it was the most secure and up-to-date one around, guaranteed. A quick perusal of mitre showed the version in question – not sure now if it was 5.3.11 – had about a half-dozen exploits. The user was completely oblivious to this.

      You know who wasn't? The team paid full-time to patch and test around the exploits on the packages they support. The PHP version available from the distro was fully-patched for all applicable exploits affecting that version. It even covered the half-dozen ones that would have laid the 'new, thus more secure' version wide open.

      I think the first thing a responsible sysadmin should do is to not randomly compile in this week's code, breaking natural upgrade potential from professionals and compatibility with the OS, and maybe trust that a team of people whose job it is to keep their stuff secure may be more proficient at it. But, YMMV if you happen to have a large team dedicated to security alerts and code rebuilds in response.

      Old base-releases of software aren't just for sadistic sport; they're for compatibility and certification for ISVs, or at least those who can code toward a firm target. "Did the brochure mention it was certified on that OS?" is a question we need to ask more often.

      Link

      - Elton Lockhart August 21, 2012, 10:02 am

        This is a great comment! Other readers should take heed of what is said here.

        Packages in a distro are patched for exploits and administration of your own compiled PHP version takes a lot of time.

        Link

      - Cody October 31, 2012, 2:32 am

        Exactly. bish is spot on. That goes even for removing files instead of commenting out the related parts. If you remove the file or rename it without having a file with the original name, there's potential for a completely new configuration file on an update. Look up rpmnew files (google or whatever). That new files could be installed would be bad in many ways including services being broken and also security issues. And if you want to comment out every line entirely, you could simply do something like this (assume comment char is # which most often is):

        sed -i 's/^/#/g' filename

        Jonathan: it's called a backport. Note how CentOS 6.3 has php 5.3.3. But do you actually think that means the source is only of php 5.3.3? Not at all. Since you posted that there's been more than 9 updates and 9 of those include one (often more than one) fix(es) to security flaws (yes, I am referring to php in CentOS). Your suggestion is therefore quite invalid. Check this:

https://access.redhat.com/security/updates/backporting/?sc_cid=3093

Note also that your thinking would also mean that if someone changes a banner to show something different than what the program is, then it must be so. And even disabling software version display doesn't necessarily mean the version is completely hidden.

Besides taking more time updating things, there's other reasons compiling is not at all appealing for production servers.

As I said earlier, package based distros will typically backport the fixes, anyway. A perfect example is one that is mentioned in this article: CentOS. Observe how CentOS 6.x is still in the 2.6.x kernel tree and the current kernel tree is 3.6 and current stable is 3.6.4. Guess what though? CentOS still update the packages when there is a need (and there's often enough security fixes included). If you want the absolute latest then go for a distro that has newer versions but observe that newer versions can equate to : new bugs, new compatibility issues, new problems in general. Also, as for, say Fedora versus CentOS, CentOS end of life time (10 years) is much longer than Fedora (2 years if I recall) and that means updates are applied longer, too.

The fact there's package based distros is a blessing and I not only love programming, I have used the distros that were built entirely from source and similar (as in Linux from scratch and Gentoo). I even had at one point worked on my own distro (and I don't mean rebranding) and while fun it would be insane to take on my own if I were to keep it updated and stable. And that last point is why binary distros are beneficial. Compatibility, (programs built for one version of a library may very well be useless if the library is not that version), stability (and what is the point of a server if it is unstable?) and much more.

And another thing that I don't think was said (if so I didn't read it as such). Why would anyone consider unpackaged programs along with packaged programs? Sure, there is –prefix and such, but that can lead to other issues as well as more to maintain and update. Yes it's possible and yes there's alternatives, but if you don't consider those alternatives it equates to more complication which leads to more issues. Package based distros help with integrity and also file clashes.

Think about this, too: humans are not perfect and no matter what you do, you will at times make mistakes. That means programmers do too, including nasty bugs that effect stability and also security (and other things). This happens a lot. I myself am guilty of making very stupid mistakes in a project of mine that caused stability issues. It was a time I should not have been working on any programming (way too tired) but did not even realize that. End result: heap corruption which is a real pain to track down. Thankfully I did get it sorted quickly but it was still ugly.

Link

- Admir Trakic November 29, 2011, 1:53 pm

  I would not trust any of mentioned commands for tracing phpbackdoors, since scripts can be also created within phpencoders and thus hard to be identifyed.
  As allways, I surely would trust outbound/indbound monotor or intrustion system of any kind no matter what.

  Link
- Blagomir Ivanov November 30, 2011, 2:18 pm

  This is very useful post. But what about securing the users in chroot environment, not VPS as OpenVZ? Does this mean that I must have separate php process running for every one of the websites hosted on the same server?

  I mean, if I start php-cgi process with spawn-fcgi, it will be started with user "website1″. But this user does not have access to files in website2, so it can not read/write/execute php files from website2. Am I right?

  Link
- Thomas December 6, 2011, 8:15 pm

  I'm also looking forward for a good sendmail wrapper (with logging capabilites).
  All i tryed skip the attachment files when using PHP mail function and a wrapper (php or sh). Any idea please ?

  Link
- Jack Wade December 7, 2011, 6:10 pm

  The best way of securing PHP is to use suPHP or Apache-mod-itk (requires apache to run as root, so it can fork/setuid to the website user to run the scripts) to run the scripts as the user who owns the specific website, rather than the web server user and also to disable allow_url_include (this makes it impossible to use include() or require() to execute external sources yet still allows file_get_contents()/readfile() and others to grab external legitimate files, like RSS feeds and so on)

  Additionally, 99% of PHP vulnerabilities are the result of bad "programmers", not server security issues.

  Link
- Aziz December 11, 2011, 2:01 am

  Nice writeup Vivek. Whats your opinion on running an apache server on fastcgi/fpm/suhosin. From a security standpoint, is suexec necessary, or applicable, in this kind of setup? Or is it sufficient to create a separate fpm pool with a non-priveleged user for each vhost?

  Link
- Guido Iaquinti December 21, 2011, 8:22 pm

  Very good job, thanks for sharing!

  Link
- Balaji December 22, 2011, 11:55 am

  Excellent sharing. Thanks. Expecting more. :)

  Link
- JIEM February 23, 2012, 5:34 am

  Hello,

  Somebody, can tell me how to disable eval() on php?
  I have add this functionson disable_functions but not working…

Link
- Yunus February 23, 2012, 5:45 am

  Make sure you have disabled it in proper php.ini file for instance Debian/Ubuntu have different ini files for cli, cgi and apache.

  Link
  - JIEM February 23, 2012, 9:08 am

    Dear Yunus,

    I have add on disable_functions in php.ini but not working..
    i'm using centos + cpanel.. can you help me?

    Link
    - Yunus February 23, 2012, 9:21 am

      sure I will try to help, pls contact me at yunus[at] bridgeinfomatics.com

      Link

- Dhanu March 24, 2012, 1:07 pm

  #5: Log All PHP Errors .

  Thanks for the source code :D

  Link
- iyrag May 21, 2012, 5:02 pm

  mysql_escape_string() will be deprecated; PDO is suggested instead :)

  Link
- Lukasz June 13, 2012, 12:58 pm

  thx man

  Link
- Satheesh July 17, 2012, 7:11 am

  it is very useful information.. thanks for cyberciti.biz

  Link
- Just a user July 25, 2012, 7:33 pm

  Thank you.
  you have no idea how much you helped me !
  cyberciti.biz is now in my top bookmarks

  btw for SQL mysql_escape_string() is not so good.
  we should all use parameter binding(for sql) and htmlpurifier for filtering user input

  Link
- zingerx August 26, 2012, 12:00 pm

  Thank you…

  Link
- Leonardo Embon August 31, 2012, 1:35 pm

  I've found the article GREAT. I have just a question. Which user should be used to deploy files to the site, where the deployer doesn't have root access? Should I "su apache" or is it better to add my user to apache group and have 775 in the whole tree?

  Link
  - James Rhys November 30, 2012, 5:33 pm

    Ideally you'll have a 'deployer' user (I've seen installations that use Jenkins or plain bash use other usernames to obfuscate the deployment user) that has write access on the directories you wish to update (/var/www/*) but only has firewall access from within your VPN/Lan (e.g – 10.10.10.101). They'll need read and write (but not execute!) permissions on the directories they deploy to.

    Link

- olashile October 19, 2012, 11:36 pm

  do you sell shell php uploaded/hosted to upload bank script& page.

  Link
- zZz December 7, 2012, 8:38 pm

  eval() – Evaluate a string as PHP code. This is often used by an attacker to hide their code and tools on the server itself. You can configure php to disable eval().

  eval() is PHP language construction.Not a PHP function.
  You can't disable it using standart disable_functions directive in your php.ini

  Use suhosin it has:

  suhosin.executor.disable_eval = On

  directive.

Thanks very nice material.Keep up.

Link

- Gustavo December 14, 2012, 7:21 pm

  Disable .htaccess support and use apache rules directly in virtual host, you obtain better performance. Set error pages.

  Disable Directory Listing
  Options -Indexes

  Error Pages
  http://httpd.apache.org/docs/2.2/custom-error.html

  Link
- ramiro April 3, 2013, 8:07 pm

  Wonderful information you share. Thanks, this has been very useful to me.

  Link
- Chris December 13, 2013, 2:30 pm

  I've also come up with a tool that checks the php.ini configuration currently in use and returns warnings/errors it finds:

  Iniscan – https://github.com/psecio/iniscan

  Could be helpful to some out there.

  Link
- Caspel December 19, 2013, 8:13 am

  Why parse_ini_file() should be disabled?
  I can't find any security issue usind this function.
  Can you explain me why?

  Link
- Mark Finzel February 16, 2014, 5:32 am

  What would your recommendation be for the open_basedir setting? I have seen conflicting reports. Do you want to do /home/user/public_html
  or just /home/user
  I have also seen mention of /home/tmp
  I have not seen a clear explanation of how to set it to avoid issues down the road.
  (I am using FastCGI so have to set in each user's php.ini file)

  Link
  - Chris Cornutt February 17, 2014, 12:33 am

    Personally, I'd suggest two things when it comes to open_basedir:

    1. Keep it inside the base of the application. This isn't the same thing as the document root as you could have files outside of that that relate to the application. I'd *never* allow access to a user's home directory though.

    2. Keep it as limited as possible. Don't specify something like "/var/www" when "/var/www/site-name/public" will work.

    Link
- Mk February 22, 2014, 1:09 pm

  1)I'm confused with open_base_dir and upload_tmp_dir. I think that upload_tmp_dir should not be inside open_basedir. Why code execution should happen in the upload directory? Currently I have set it outside open_basedir and I have only a warning when uploading images in an installation of wordpress because it tries to determine the type of file inside the upload directory which I think it shouldn't try to do this in the first place. I'm still studying that though. Everything else works correctly including file upload plugins etc…

  2)I have the same question about session.save_path. Should code execution in session.save_path be allowed?

  3)Also Is it correct for session.save_path and upload_tmp_dir to share the same directory?

  4) And finally does anybody know what folder permmisions should the session.save_path and upload_tmp_dir directories have? I think that the answer on this is 600 but I would like to know what others think.

  Link
  - Chris Cornutt February 23, 2014, 7:47 pm

    Some answers…

    1) The way that PHP operates, upload_tmp_dir *has* to be inside the open_basdedir if it's set. Otherwise it cannot write to the directory when a user uploads the file. Keep in mind, though, that you should be immediately moving the temporary file once the user uploads it. Also remember, you can have multiple directories for open_basedir seperated with a colon (:).

    2) Not sure what you mean by "code execution" but I assume by your first question, you mean the open_basedir setting. As far as I know, session save path is influenced by open_basedir so it'd need to be included in the open_basedir list as well.

    3) Personally, I'd recommend against the save_path and upload_tmp_dir sharing the same path. It'd be better for a separation of concerns to have them in different places.

    4) The permissions depend on who owns the directory, really. If you set the owner to the web server user, then 600 should be fine (you might need 700, not sure).

Link

- Cody March 13, 2014, 5:57 pm

  And to elaborate on point 4 (not that I think Chris did any thing incorrect or wrong here – I just feel there's no such thing as too much information on the subject of permissions). Also, apologies if I went overboard with some of this but I got carried away with explaining modes and didn't realise how far I went until after the fact. Anyway:

  As Chris wrote it depends on who owns the directory and more so what permission is needed. But – and here's the important bit (pardon the pun… permission bits and all) – use the most restrictive permission as possible, always (okay, obviously you shouldn't mess around with most of your directories, binaries, libraries, etc., and you should never blindly change ownership of these without knowing what you're doing – chmod and chown can be very dangerous especially when recursively operating but for /var/www or wherever you have your web directory you should be fine, especially if you don't recursively chmod [in general the -R option is something you need to be careful of, for chmod and chown]). So if you can get away with 400 then by all means do so (I doubt you'd be OK with that since then you'd have to be root to write to the files, but … the point remains the same: restrict where you can as much as you can, as long as it is safe). On the subject of whether read is sufficient, there's also the possibility that – for example – you have a user (regular) for editing files in the web directory (as a virtual host, say) and the apache user is the group of the directory (for example it has a CMS that might update a file) or alternatively, if you don't have a CMS, you have the group also be the same as the owner and only allow read/execute for the rest of the world (so that apache can open and serve files to viewers of the website(s)). Then, you can just grant rwx to the user and apache (or just others) can get away with rx (so can read files and can open directory). Others (world) should never include write on a website. If there were any files that the site itself (let's say a CMS) needs to edit (say on update of the CMS) then you can grant those files write access to the apache user (or however you have apache set up). In short: you're restricting it as much as possible but not breaking anything either.

  An important elaboration on permissions is it also goes for database permissions. Example: wordpress documentation insists you need to GRANT ALL to the user on the database that wordpress uses. However, that is complete and utter nonsense. I know because I use much more restricted on my wordpress installs and there has never been a problem related to this (mod_security2 and other things can cause issues but that's nothing to do with the database itself and mod_security2 is actually well worth any hassle getting it to function correctly [false positives, certain rules needing modifications for your site, whatever it is]). Furthermore, and I seem to remember wordpress documentation is guilty here too, when people suggest that to fix (or make sure it isn't a permissions issue) read/write permissions you should (even if temporarily) set 777, don't do it. For your own sake and for your system's sake (why on earth would anyone think a directory should be world read/write/execute to make sure there is no permission issues, is to this day something that bewilders and bemuses me). The only directories that should be 777 (which are actually 1777 – restricted delete; 1777 on a file would be sticky bit with read/write/execute) are directories like /var/tmp and /tmp

  If you ever do need write and read permission, then 6 (or rw) should suffice. If you only need read then 4 suffices. Likewise, if you only need read and execute, then 5 suffices. Also, as for execute bit (1=x, 2=w, 4=r, and its bits so binary which is why 7 = rwx, 6 = rw, 5 = rx, etc. Note: the modes itself, that is 0-7, is in fact not binary but octal [hence 0-7] but they add up like binary [observe that 8 is divisible by 2 just like 16 is, which is why octal and hexadecimal are often used in programming – much more convenient than binary), it depends on if the file is a directory or a regular file as to what it is used for. For regular files it allows execution. For directories it allows: changing to the directory (you'll need +r for viewing files in it and +w for writing to the directory).

  Any way, hopefully that is of value to _someone_ (and hopefully the fact I am still trying to wake up did not allow me to make any stupid mistakes/errors but if I did I am sure someone will correct me – I hope so anyway)!

  Link

- Donald December 31, 2014, 3:16 am

  why no mention of suphp?

  Link

- Bhumi March 5, 2015, 12:18 pm

  Useful. Thanks.

  Link

Next post: HowTo: Linux Update the Adobe Flash Player [Firefox and Chrome Plugin]

Previous post: Download Fedora 16 CD / DVD ISO

To search, type and hit enter

Featured Articles:

- 30 Cool Open Source Software I Discovered in 2013
- 30 Handy Bash Shell Aliases For Linux / Unix / Mac OS X
- Top 30 Nmap Command Examples For Sys/Network Admins
- 25 PHP Security Best Practices For Sys Admins
- 20 Linux System Monitoring Tools Every SysAdmin Should Know
- 20 Linux Server Hardening Security Tips
- Linux: 20 Iptables Examples For New SysAdmins
- Top 20 OpenSSH Server Best Security Practices
- Top 20 Nginx WebServer Best Security Practices
- 20 Examples: Make Sure Unix / Linux Configuration Files Are Free From Syntax Errors
- 15 Greatest Open Source Terminal Applications Of 2012
- My 10 UNIX Command Line Mistakes
- Top 10 Open Source Web-Based Project Management Software
- Top 5 Email Client For Linux, Mac OS X, and Windows Users
- The Novice Guide To Buying A Linux Laptop