PHPSecurityBestPractices

PHP is an open source server side scripting language and it is most popular and widely used. LiteSpeed or other web server provides access to content and files via HTTP or HTTPS. Web servers are publicly accessible, so they possess possible security vulnerabilities. If your PHP is not configured properly, it can create a lot of problems for you. So PHP should be configured with caution. These security techniques can be done with a few lines of code or some little adjustments to your application settings. The following are the best habits that a programmer can develop in order to configure PHP securely.



## Table of Contents

1. SQL Injection
2. Keep PHP, Software and OS Updated
3. Restrict File and Directory Access
4. Cross-site scripting (XSS)
5. Log All PHP Errors
6. Disallow Uploading Files
7. Document Root Setup
8. Cross-Site Request Forgery (CSRF)
9. PHP Security Tips
10. Conclusion

## #1 SQL Injection

SQL injection is the most commonly used hacking technique used by hackers to interact with the database side of a web application. Through SQL injection, attackers make full use the vulnerabilities of web application security and send unsafe queries to the database. These queries can modify or alter database tables or delete the whole database. It is a vulnerability in the database layer of a PHP application. When user input is incorrectly filtered, any SQL statement can be executed by the application.

If user input is inserted without modification into an SQL query, then the application becomes vulnerable to SQL Injection, like in the following example:

$unsafe = $\_POST['input'];

mysql\_query("INSERT INTO `table` (`column`) VALUES ('$unsafe')");

That’s because the user can input something like value'); DROP TABLE table;, and the query becomes:

INSERT INTO `table` (`column`) VALUES('value'); DROP TABLE table;')

### How to Prevent this:

Use prepared statements and parameterized queries. These are SQL statements that are sent to and parsed by the database server separately from any parameters. This way it is impossible for an attacker to inject malicious SQL.

Using PDO (for any supported database driver):

$stmt = $pdo->prepare('SELECT \* FROM employees WHERE name = :name');

$stmt->execute(array('name' => $name));

foreach ($stmt as $row) {

// do something with $row

When using PDO to access a MySQL database real prepared statements are not used by default. To fix this you have to set PDO::ATTR\_EMULATE\_PREPARES, false. Create a connection using PDO like this:

$dbConnection = new PDO('mysql:dbname=dbtest;host=127.0.0.1;charset=utf8', 'user', 'pass');

$dbConnection->setAttribute(PDO::ATTR\_EMULATE\_PREPARES, false);

$dbConnection->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

## #2 Keep PHP, Software and OS Updated

Keeping your PHP, Software and operating system up to date is an important part of making your web application secure. PHP 7.3 is the most secure and stable version. I recommend you upgrade your web application to the newest version of PHP. If you’re installing PHP via a package manager, you can use the following commands:

# yum update

OR

# apt-get update && apt-get upgrade

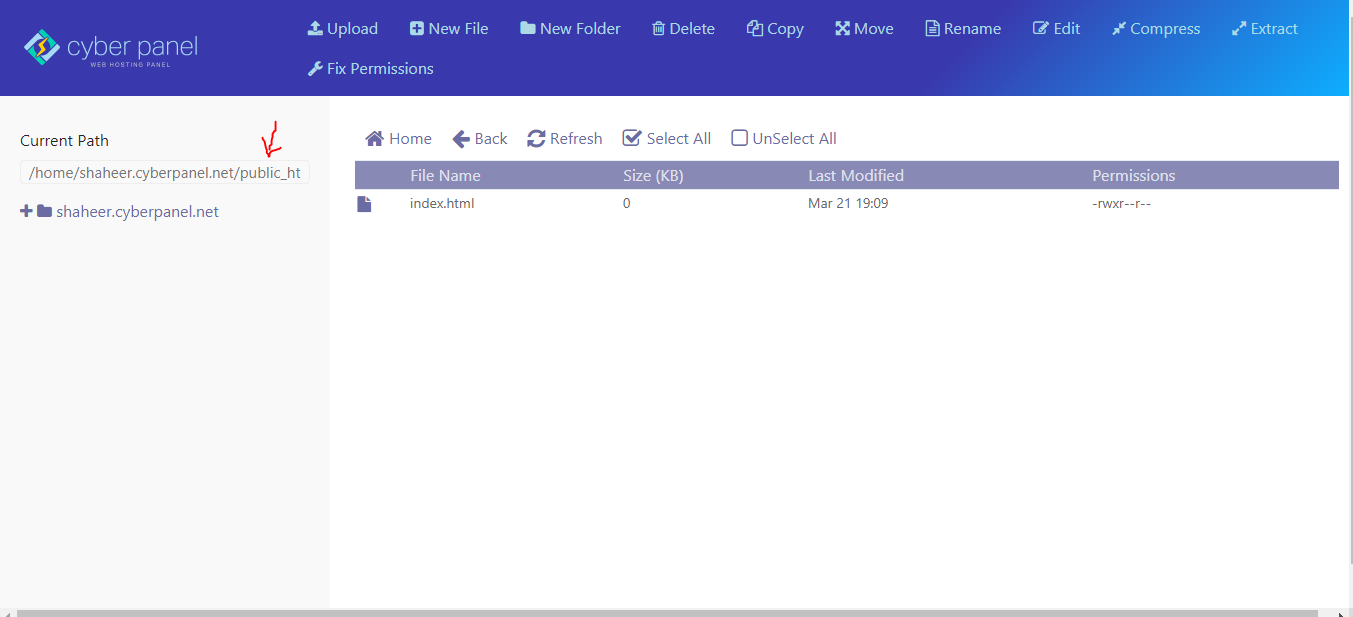
Incase if you are on CyberPanel, you can just use your package manager to upgrade your packages and it will also upgrade PHP related packages if any upgrades are available.

## #3 Restrict File and Directory Access

Make sure your files and DocumentRoot are owned by a non-root user under /home/website\_name/public\_html.

/home/website\_name/public\_html is a subdirectory, and DocumentRoot is modifiable by other users. Root never executes any files out of there, and shouldn’t be creating files in there.

You can see the CyberPanel DocumentRoot in an image below:



Make sure file permissions are set to 0644 (read-only) under /home/website\_name/public\_html:

# chmod -R 0644 /home/shaheer.cyberpanel.net/public\_html/

Make sure all directories permissions are set to 0755 under /home/website\_name/public\_html:

# find /home/shaheer.cyberpanel.net/public\_html/ -type d -print0 | xargs -0 -I {} chmod 0755 {}

## #4 Cross-site scripting (XSS)

XSS – Cross-site scripting – is a vulnerability in PHP web applications, which attackers may exploit to steal a user’s information. The actual attack on the victim happens when the end user visits the page and some unsafe/malicious code executes.  Write more secure PHP scripts (validating all user input) to avoid XSS attacks.

Cross-site scripting may also be used to disfigure the web application instead of targeting the end users.

### How to Prevent XSS

To keep yourself protected from XSS attack, you must disinfect your input. Your PHP web application code should never output data received as input directly to the browser without checking it for unsafe/malicious code.

## #5 Log All PHP Errors

Make sure your PHP error file message is not shown to all site visitors. You can use CyberPanel [advanced PHP Editor](https://cyberpanel.net/docs/2-php-configurations/) for making these changes.

display\_errors=Off

Keep in mind PHP gives you a lot of information about the paths, database directories and other sensitive information. I recommend that you put error logging in place to show the error on your website.

log\_errors=On

error\_log=/var/log/httpd/php\_scripts\_error.log

## #6 Disallow Uploading Files

I strongly advise you to edit PHP settings using CyberPanel [advanced PHP Editor](https://cyberpanel.net/docs/2-php-configurations/) and set the following directive to disable file uploads for security reasons:

file\_uploads=Off

If your client want to upload a file to your PHP application, turn this feature on upload\_max\_filesize by going on the PHP configuration file So that the PHP accept the limit size of files through upload.

file\_uploads=On

# user can only upload upto 2MB via php upload file

upload\_max\_filesize=2M

## #7 Document Root Setup

Document Root is the folder or directory where your PHP web application files for a domain name have been saved. Your Document Root must be set to /home/example.cyberpanel.net/public\_html Where example.cyberpanel.net is the domain name which was configures automatically by CyberPanel. public\_html serves the index.php page in your PHP web application. The purpose of making the public\_html directory hide the sensitive files like .htaccess and the error log file in your domain name directory is so that it will not be accessible publicly to anyone. I strongly recommend you not save your sensitive files to your root directory. You can make a new directory to save those files.

## #8 Cross-Site Request Forgery (CSRF)

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email/chat), an attacker may force the users of a web application to execute actions of the attacker’s choosing. A successful CSRF exploit can compromise end user data and operation in the case of a normal user. If the targeted end user is the administrator account, this can compromise the entire web application.

### Procedures to prevent CSRF

Checking if the request has a valid session cookie is not enough. We need to check if a unique identifier is sent with every HTTP request sent to the application. CSRF requests WON’T have this valid unique identifier. The reason CSRF requests won’t have this unique request identifier is the unique ID is rendered as a hidden field on the page and is appended to the HTTP request once a link/button press is selected. The attacker will have no knowledge of this unique ID, as it is random and rendered dynamically per link, per page.

1. A list is complied prior to delivering the page to the user. The list contains all valid unique IDs generated for all links on a given page. The unique ID could be derived from a secure random generator such as SecureRandom for J2EE.
2. A unique ID is appended to each link/form on the requested page prior to being displayed to the user.
3. Maintaining a list of unique IDs in the user session, the application checks if the unique ID passed with the HTTP request is valid for a given request. If the unique ID passed with the HTTP request is valid for a given request.
4. If the unique ID is not present, terminate the user session and display an error to the user.

## Conclusion

After following these steps your PHP web application will be solidified. You’ll be all done and may run your PHP web pages. However be aware of vulnerabilities and malicious activities of attackers caused by not using the best practice programming rules. Many good companies also use tools to check the security vulnerabilities in PHP web applications. I have tried to cover some PHP security issues. There are a lot of other ways as well to make your PHP application secure. You should consult further resources for your web application’s security.

<https://cyberpanel.net/blog/2019/03/29/php-security-best-practices/>