How to secure PHP web applications and prevent attacks?

9 min read · Last change January 21, 2018

As a developer, you must know how to build secure and bulletproof applications. It is your duty is to ensure the security of your applications and to prevent attacks.

Checklist of PHP and web security issues

Make sure you have these items sorted out when deploying your applications to production environments:

1. Cross site scripting (XSS)
2. Injections
    - SQL injection
    - Directory traversal (path injection)
    - Command injection
    - Code injection
3. Cross site request forgery (XSRF/CSRF)
4. Public files
5. Passwords
6. Uploading files
7. Session hijacking
8. Remote file inclusion
9. PHP configuration
    - Error reporting
    - Exposing PHP Version
    - Remote files
    - open_basedir
    - Session settings
10. Use HTTPS
11. Things not listed

Cross site scripting (XSS)

XSS attacks happen when client-side code (usually JavaScript) gets injected into the output of your PHP script. This can be through the URL, but can also occur via a stored technique such as the database.

```
// GET data is sent through URL: http://example.com/search.php?search=<script>alert('test')
</script> $search = $_GET['search'] ?? null; echo 'Search results for '.$search; // This can be
solved with htmlspecialchars $search = htmlspecialchars($search, ENT_QUOTES, 'UTF-8'); echo
'Search results for '.$search;
```

- `ENT_QUOTES` is used to escape single and double quotes beside HTML entities
- UTF-8 is used for pre PHP 5.4 environments (now it is default). In some browsers some characters might get pass the `htmlspecialchars()`.

Injections

SQL injection

When accessing databases from your application, SQL injection attack can happen by injecting malicious SQL parts into your existing SQL statement.

- More details available in "What is SQL injection and how to prevent it?" FAQ.

Directory traversal (path injection)

Directory traversal attacks, also known as `../` (dot, dot, slash) attacks, happen when users supply filenames as input that can traverse to parent directories. Data can be set as `index.php?page=../secret`, or `/var/www/secret`, or something more catastrophic:

```
$page = $_GET['page'] ?? 'home'; require $page; // or something like this echo
file_get_contents('../pages/'.$page.'.php');
```

In such cases you must check if there are attempts to access the parent or some remote folder:

```
// Checking if the string contains parent directory if (strstr($_GET['page'], '../') !== false) {
throw new \Exception("Directory traversal attempt!"); } // Checking remote file inclusions if
(strstr($_GET['page'], 'file://') !== false) { throw new \Exception("Remote file inclusion
attempt!"); } // Using whitelists of pages that are allowed to be included in the first place
$allowed = ['home', 'blog', 'gallery', 'catalog']; $page = (in_array($page, $allowed)) ? $page :
'home'; echo file_get_contents('../pages/'.$page.'.php');
```

Command injection

Be careful when dealing with commands executing functions and data you don't trust.

```
exec('rm -rf '.$GET['path']);
```

Code injection

Code injection happens when malicious code can be injected via the `eval()` function, so remember to always sanitize your data when using it:

```
eval('include '.$_GET['path']);
```

Cross site request forgery (XSRF/CSRF)

Cross site request forgery, one click attacks, or session riding is an exploit whereby users execute unwanted actions on web applications.

Public files

Make sure to move all your application files, configuration files and similar parts of your web application to a folder that isn't publicly accessible when you visit URLs of your web application. Some types of files (e.g., `.yml` files) might not be processed by your web server and users could view them online.

An example of good folder structure:

```
app/ config/ parameters.yml src/ public/ index.php style.css javascript.js logo.png
```

Configure your web server to serve files from the `public` folder instead of from your application root folder. The public folder contains the front controller (`index.php`). In case of a web server misconfiguration resulting in PHP files failing to be served properly, the source code of `index.php` will be visible to the public.

- More details are available in the dedicated FAQ: How to use configuration in PHP applications?

Passwords

When working with users' passwords, hash them properly with the `password_hash()` function.

- More details are available in "How to work with users' passwords and how to securely hash passwords in PHP?" FAQ.

Uploading files

Many security breaches occur when users can upload files onto a server. Make sure you go through all the vulnerabilities associated with uploading files and take appropriate precautions against these vulnerabilities, such as by renaming uploaded files, moving them to publicly inaccessible folders, checking the types of files uploaded and so on. Since there are many issues to check here, more information is also located in the separate FAQ:

- How to securely upload files with PHP? FAQ.

Session hijacking

Session hijacking is an attack where an attacker steals the session ID of a user. The session ID is sent to the server where the associated `$_SESSION` array is populated. Session hijacking is possible through an XSS attack or when someone gains access to the folder on a server where the session data is stored.

Remote file inclusion

An RFI (remote file inclusion) attack is when an attacker can include custom scripts:

```
$page = $_GET['page'] ?? 'home' require $page . '.php';
```

In the above code, `$_GET` can be set to a remote file `http://yourdomain.tld/index.php?page=http://example.com/evilscript`

Make sure you disable this in your `php.ini` unless you know what you're doing:

```
; Disable including remote files allow_url_fopen = off ; Disable opening remote files for include(), require() and include_once() functions. ; If above allow_url_fopen is disabled, allow_url_include is also disabled. allow_url_include = off
```

PHP configuration

Always keep the installed PHP version updated. You can use versionscan to check for possible vulnerabilities of your PHP version. Update open source libraries and applications, and keep your web server well maintained.

Here are some of the important settings from `php.ini` that you should check out. You can also use iniscan to scan your `php.ini` files for best security practices.

Error reporting

In your production environment, you must always turn off displaying errors to the screen. If errors occur in your application and they are visible to the outside world, an attacker could get valuable data for attacking your application. `display_errors` and `log_errors` directives in the `php.ini` file:

```
; Disable displaying errors to screen display_errors = off ; Enable writing errors to server logs log_errors = on
```

- More information in chapter Errors.

Exposing PHP version

PHP version is visible in HTML headers. You might want to consider hiding your PHP version by turning off the `expose_php` directive, preventing the web server from sending back the `X-Powered-By` header:

```
expose_php = off
```

Remote files

In most cases, it's important to disable access to remote files:

```
; disabled opening remote files for fopen, fsockopen, file_get_contents and similar functions
allow_url_fopen = 0 ; disabled including remote files for require, include ans similar functions
allow_url_include = 0
```

open_basedir

This settings defines one or more directories (subdirectories included) where PHP has access to read and write files. This includes file handling (`fopen`, `file_get_contents`) and also including files (`include`, `require`):

```
open_basedir = "/var/www/test/uploads"
```

Session settings

- **session.use_cookies** and **session.use_only_cookies**

  PHP is by default configured to store session data on the server and a tracking cookie on client-side (usually called `PHPSESSID`) with unique ID for the session.

```
; in most cases you'll want to enable cookies for storing session session.use_cookies = 1 ;
disabled changing session id through PHPSESSID parameter (e.g foo.php?PHPSESSID=<session id>)
session.use_only_cookies = 1 session.use_trans_sid = 0 ; rejects any session ID from user that
doesn't match current one and creates new one session.use_strict_mode = 0
```

- **session.cookie_httponly**

  If the attacker somehow manages to inject JavaScript code for stealing a user's current cookies (the `document.cookie` string), the `HttpOnly` cookie you've set won't show up in the list.

```
session.cookie_httponly = 1
```

- **session.cookie_domain**

  This sets the domain for which cookies apply. For wildcard domains you can use `.example.com`, or set this to the domain where it should be applied. By default, it isn't enabled, so it's highly recommended for you to enable it:

```
session.cookie_domain = example.com
```

- **session.cookie_secure**

  For HTTPS sites, this accepts only cookies sent over HTTPS. If you're still not using HTTPS, you should consider it.

```
session.cookie_secure = 1
```

Use HTTPS

HTTPS is a protocol for securely communication over networks. It's highly recommended that you enable it on all sites. Read more about HTTPS in the dedicated FAQ: How to install an SSL certificate and enable HTTPS.

What is next?

Above we've introduced many security issues. Security, attacks, and vulnerabilities are continuously evolving. Take your time and read through some good resources to learn more about security and turn this check list into a habit:

- General:
  - Awesome AppSec - A curated list of resources for learning about application security.
  - OWASP - The Open Web Application Security Project, organization focused on improving security of software.
  - Security Guide for Developers
  - The Basics of Web Application Security, by Martin Fowler.
- PHP focused:
  - PHP Manual - A must read security chapter in official documentation.
  - Codecourse videos - Demos and advice on the most common PHP security areas.
  - DVWA, Damn Vulnerable Web Application - Example of unsecure web application to test your skills and tools.
  - OWASP PHP Security Cheat Sheet - Basic PHP security tips for developers and administrators.
  - Securing PHP - Website and books with basic topics and specific cases in authentication/authorization and exploit prevention.
  - SensioLabs Security - SensioLabs Security Advisories Checker for checking your PHP project for known security issues
  - The most forgotten web vulnerabilities - Recommended PDF article.
  - websec.io - Dedicated to educating developers about security with topics relating to general security fundamentals, emerging technologies and PHP-specific information.
- Tools:
  - iniscan - A php.ini scanner for best security practices.
  - Kali Linux - Penetration testing Linux distribution.
  - Observatory by Mozilla - Online security checker.
  - versionscan - PHP version scanner for reporting possible vulnerabilities.
  - Roave Security Advisories - This package ensures that your application doesn't have installed dependencies with known security vulnerabilities.
  - WebSecTools - List of useful web security related tools.
  - OWASP Zed Attack Proxy - Free security tool, available also on GitHub.

Security

How to secure PHP web applications and prevent attacks? How to work with users' passwords and how to securely hash passwords in PHP? What is SQL injection and how to prevent it? How to securely upload files with PHP? Configuration in PHP applications How to protect and hide PHP source code? How to install an SSL certificate and enable HTTPS? Encryption, hashing, encoding and obfuscation

Found a typo? Something wrong with this content?

Just fork and edit it.

About PHP.earth

Sitemap Team Get Involved Status

PHP.earth documentation

Index PHP installation wizard FAQ <?php tips

Community

Facebook Group GitHub Slack Twitter