

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- COBOL
- C#
- CSS
- Flex
- Go
- HTML
- Java
- JavaScript
- Kotlin
- Objective C
- PHP
- PL/I
- PL/SQL
- Python
- RPG
- Ruby
- Scala
- Swift
- Terraform
- Text
- TypeScript
- T-SQL
- VB.NET
- VB6
- XML



PHP static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your PHP code

All rules 268

Vulnerability 40

Bug 51

Security Hotspot 33

Code Smell 144

Tags ▾

Search by name... 🔍

enabled
<div>Vulnerability</div>
"allow_url_fopen" and "allow_url_include" should be disabled
<div>Vulnerability</div>
"open_basedir" should limit file access
<div>Vulnerability</div>
Neither DES (Data Encryption Standard) nor DESede (3DES) should be used
<div>Vulnerability</div>
"exit(...)" and "die(...)" statements should not be used
<div>Bug</div>
Functions and variables should not be defined outside of classes
<div>Code Smell</div>
Track lack of copyright and license headers
<div>Code Smell</div>
Octal values should not be used
<div>Code Smell</div>
Switch cases should end with an unconditional "break" statement
<div>Code Smell</div>
Session-management cookies should not be persistent
<div>Vulnerability</div>
Cryptographic RSA algorithms should always incorporate OAEP (Optimal Asymmetric Encryption Padding)
<div>Vulnerability</div>
SHA-1 and Message-Digest hash algorithms should not be used in

Setting loose POSIX file permissions is security-sensitive

Analyze your code

Security Hotspot

Major ?

cwe sans-top25 owasp

In Unix, "others" class refers to all users except the owner of the file and the members of the group assigned to this file.

Granting permissions to this group can lead to unintended access to files.

Ask Yourself Whether

- The application is designed to be run on a multi-user environment.
- Corresponding files and directories may contain confidential information.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

The most restrictive possible permissions should be assigned to files and directories.

Sensitive Code Example

```
chmod("foo", 0777); // Sensitive
```

```
umask(0); // Sensitive
umask(0750); // Sensitive
```

For [Symfony Filesystem](#):

```
use Symfony\Component\Filesystem\Filesystem;

$fs = new Filesystem();
$fs->chmod("foo", 0777); // Sensitive
```

For [Laravel Filesystem](#):

```
use Illuminate\Filesystem\Filesystem;


$fs = new Filesystem();
$fs->chmod("foo", 0777); // Sensitive
```

Compliant Solution

```
chmod("foo", 0750); // Compliant
```

```
umask(0027); // Compliant
```

algorithms should not be used in secure contexts

 Vulnerability

Assertions should not be made at the end of blocks expecting an exception

 Bug

Regular expressions should be syntactically valid

 Bug

Only one method invocation is expected when testing exceptions

 Bug

Reading the Standard Input is security-sensitive

For [Symfony Filesystem](#):

```
use Symfony\Component\Filesystem\Filesystem;

$fs = new Filesystem();
$fs->chmod("foo", 0750); // Compliant
```

For [Laravel Filesystem](#):

```
use Illuminate\Filesystem\Filesystem;

$fs = new Filesystem();
$fs->chmod("foo", 0750); // Compliant
```

See

- [OWASP Top 10 2021 Category A1](#) - Broken Access Control
- [OWASP Top 10 2021 Category A4](#) - Insecure Design
- [OWASP Top 10 2017 Category A5](#) - Broken Access Control
- [OWASP File Permission](#)
- [MITRE, CWE-732](#) - Incorrect Permission Assignment for Critical Resource
- [MITRE, CWE-266](#) - Incorrect Privilege Assignment
- [SANS Top 25](#) - Porous Defenses

Available In:

sonarcloud  | **sonarqube** 