



C#

3 CSS

 $\bowtie$ Flex

-GO

Go 5 HTML

Java

JavaScript JS

Kotlin

Objective C

**PHP** 

PL/I

PL/SQL

Python

RPG

Ruby

Scala

N. Swift

Terraform

Text 月

TS 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) 6 Vulnerability (40)

**R** Bug (51)

Tags

**Caught Exceptions must** 

derive from Throwable

**Noncompliant Code Example** 

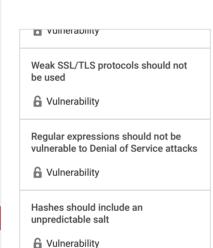
class NoThrowable {}

Security Hotspot 33

Search by name...

Code Smell (144)

Analyze your code



Regular expressions should have valid

Rug Bug

delimiters

Regex lookahead assertions should not be contradictory

Rug Bug

Back references in regular expressions should only refer to capturing groups that are matched before the reference

Rug Bug

Regex boundaries should not be used in a way that can never be matched

Regex patterns following a possessive quantifier should not always fail

Assertion failure exceptions should not be ignored

👬 Bug

References used in "foreach" loops should be "unset"

Using clear-text protocols is securitysensitive

```
👬 Bua 🔒 Blocker 🚱
                               unused
Instances of classes that do not derive from the "Throwable" interface
cannot be used in a PHP "throw" statement. Thus, it does not make sense to
try to catch such objects within a "try-catch" block.
Many built-in exceptions such as "Exception" and the SPL exception classes
do implement the "Throwable" interface and can be extended when creating
custom exceptions.
This rule raises an issue when the classes used to specify the type of objects
to be caught in a "try-catch" block do not derive from "Throwable" .
```

## **Compliant Solution**

foo();

try {

```
<?php
class SomeThrowable implements Throwable {
    // Implementation of the Throwable methods
try {
} catch (SomeThrowable $e) { // Compliant
class SomeCustomException extends Exception {}
try {
    foo();
} catch (SomeCustomException $e) { // Compliant
}{code}
```

} catch (NoThrowable \$e) { // Noncompliant

Available In:

sonarlint ⊕ | sonarcloud ↔ | sonarqube

Expanding archive files without controlling resource consumption is security-sensitive

Security Hotspot

Signalling processes is security-sensitive

Security Hotspot

Configuring loggers is security-sensitive

Security Hotspot

Using weak hashing algorithms is security-sensitive

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy