

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- 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...

Code Smell

References should not be passed to function calls

Code Smell

"switch" statements should have "default" clauses

Code Smell

Control structures should use curly braces

Code Smell

String literals should not be duplicated

Code Smell

Methods should not be empty

Code Smell

Constant names should comply with a naming convention

Code Smell

Secret keys and salt values should be robust

Vulnerability

Authorizations should be based on strong decisions

Vulnerability

Server-side requests should not be vulnerable to forging attacks

Vulnerability

The number of arguments passed to a function should match the number of parameters

Bug

Non-empty statements should change control flow or have at least one side-effect

Regular expressions should not be vulnerable to Denial of Service attacks

Analyze your code

Vulnerability

Critical

injection cwe owasp
denial-of-service

Most of the regular expression engines use backtracking to try all possible execution paths of the regular expression when evaluating an input, in some cases it can cause performance issues, called catastrophic backtracking situations. In the worst case, the complexity of the regular expression is exponential in the size of the input, this means that a small carefully-crafted input (like 20 chars) can trigger catastrophic backtracking and cause a denial of service of the application. Super-linear regex complexity can lead to the same impact too with, in this case, a large carefully-crafted input (thousands chars).

PHP prevents Denial of Service attacks with [configuration settings](#) set by default to safe values. If the `pcre.backtrack_limit` or `pcre.recursion_limit` settings are set to higher values than the default values, make sure that it is not too large numbers that will expose the application to Denial of Service in the event of incorrect or malicious regex evaluation. However, despite this mitigation it is recommended to validate/escape user-controlled inputs.

It is not recommended to construct a regular expression pattern from a user-controlled input, if no other choice, sanitize the input to remove/annihilate regex metacharacters.

Noncompliant Code Example

```
$regex = $_GET["regex"];  
$input = $_GET["input"];  
  
preg_grep($regex, $input); // Noncompliant
```

Compliant Solution

```
$regex = $_GET["regex"];  
$input = $_GET["input"];  
  
preg_grep(preg_quote($regex), $input); // Compliant
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-20](#) - Improper Input Validation
- [MITRE, CWE-400](#) - Uncontrolled Resource Consumption
- [MITRE, CWE-1333](#) - Inefficient Regular Expression Complexity
- [OWASP Regular expression Denial of Service - ReDoS](#)

 Bug

Variables should be initialized before use

 Bug

Replacement strings should reference existing regular expression groups

 Bug

Alternation in regular expressions should not contain empty alternatives

 Bug

Unicode Grapheme Clusters should be avoided inside regex character classes

Available In:

sonarcloud 

sonarqube  Developer Edition

© 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](#)