

- 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...



OS commands should not be vulnerable to argument injection attacks

Vulnerability

Logging should not be vulnerable to injection attacks

Vulnerability

Repeated patterns in regular expressions should not match the empty string

Bug

Function and method parameters' initial values should not be ignored

Bug

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive

Security Hotspot

Delivering code in production with debug features activated is security-sensitive

Security Hotspot

Creating cookies without the "HttpOnly" flag is security-sensitive

Security Hotspot

Creating cookies without the "secure" flag is security-sensitive

Security Hotspot

Using hardcoded IP addresses is security-sensitive

Security Hotspot

Regular expression quantifiers and character classes should be used concisely

Code Smell

Character classes should be preferred

Related "if/else if" statements and "cases" in a "switch" should not have the same condition

Analyze your code

Bug Major unused pitfall

A switch and a chain of if/else if statements is evaluated from top to bottom. At most, only one branch will be executed: the first one with a condition that evaluates to true.

Therefore, duplicating a condition automatically leads to dead code. Usually, this is due to a copy/paste error. At best, it's simply dead code and at worst, it's a bug that is likely to induce further bugs as the code is maintained, and obviously it could lead to unexpected behavior.

For a switch, if the first case ends with a break, the second case will never be executed, rendering it dead code. Worse there is the risk in this situation that future maintenance will be done on the dead case, rather than on the one that's actually used.

On the other hand, if the first case does not end with a break, both cases will be executed, but future maintainers may not notice that.

Noncompliant Code Example


```
if ($param == 1)
    openWindow();
else if ($param == 2)
    closeWindow();
else if ($param == 1) // Noncompliant
    moveWindowToTheBackground();

switch($i) {
    case 1:
        //...
        break;
    case 3:
        //...
        break;
    case 1: // Noncompliant
        //...
        break;
    default:
        // ...
        break;
}
```


Compliant Solution

```
if ($param == 1)
    openWindow();
```


over reluctant quantifiers in regular expressions

 Code Smell


A subclass should not be in the same "catch" clause as a parent class

 Code Smell

Jump statements should not be redundant

 Code Smell

"catch" clauses should do more than rethrow

 Code Smell

"&&" and "||" should be used

```
else if ($param == 2)
    closeWindow();
else if ($param == 3)
    moveWindowToTheBackground();
```

```
switch($i) {
    case 1:
        //...
        break;
    case 3:
        //...
        break;
    default:
        // ...
        break;
}
```

Available In:

sonarlint  | **sonarcloud**  | **sonarqube** 