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...

be used to end lines

Code Smell

More than one property should not be declared per statement

Code Smell

The "var" keyword should not be used

Code Smell

"<?php" and "<?=" tags should be used

Code Smell

File names should comply with a naming convention

Code Smell

Comments should not be located at the end of lines of code

Code Smell

Local variable and function parameter names should comply with a naming convention

Code Smell

Field names should comply with a naming convention

Code Smell

Lines should not end with trailing whitespaces

Code Smell

Files should contain an empty newline at the end

Code Smell

Modifiers should be declared in the correct order

Code Smell

An open curly brace should be located at the beginning of a line

Code Smell

Using command line arguments is security-sensitive

Analyze your code

Security Hotspot Critical

Using command line arguments is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2018-7281
- CVE-2018-12326
- CVE-2011-3198

Command line arguments can be dangerous just like any other user input. They should never be used without being first validated and sanitized.

Remember also that any user can retrieve the list of processes running on a system, which makes the arguments provided to them visible. Thus passing sensitive information via command line arguments should be considered as insecure.

This rule raises an issue when on every program entry points (`main` methods) when command line arguments are used. The goal is to guide security code reviews.

Ask Yourself Whether

- any of the command line arguments are used without being sanitized first.
- your application accepts sensitive information via command line arguments.

If you answered yes to any of these questions you are at risk.

Recommended Secure Coding Practices

Sanitize all command line arguments before using them.

Any user or application can list running processes and see the command line arguments they were started with. There are safer ways of providing sensitive information to an application than exposing them in the command line. It is common to write them on the process' standard input, or give the path to a file containing the information.

Sensitive Code Example

Builtin access to `$argv`

```
function globfunc() {
    global $argv; // Sensitive. Reference to global $argv
    foreach ($argv as $arg) { // Sensitive.
        // ...
    }
}

function myfunc($argv) {
    $param = $argv[0]; // OK. Reference to local $argv param
    // ...
}

foreach ($argv as $arg) { // Sensitive. Reference to $argv.
    // ...
}
```

An open curly brace should be located at the end of a line

Code Smell

Tabulation characters should not be used

Code Smell

Method and function names should comply with a naming convention

Code Smell

Creating cookies with broadly defined "domain" flags is security-sensitive

Security Hotspot

```
$myargv = $_SERVER['argv']; // Sensitive. Equivalent to $arg

function serve() {
    $myargv = $_SERVER['argv']; // Sensitive.
    // ...
}

myfunc($argv); // Sensitive

$myvar = $HTTP_SERVER_VARS[0]; // Sensitive. Note: HTTP_SERV

$options = getopt('a:b:'); // Sensitive. Parsing arguments.

$GLOBALS["argv"]; // Sensitive. Equivalent to $argv.

function myglobals() {
    $GLOBALS["argv"]; // Sensitive
}

$argv = [1,2,3]; // Sensitive. It is a bad idea to override
```

Zend Console

```
new Zend\Console\Getopt(['myopt|m' => 'this is an option']);
```

Getopt-php library

```
new \GetOpt\Option('m', 'myoption', \GetOpt\GetOpt::REQUIRED
```

See

- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-88 - Argument Injection or Modification
- MITRE, CWE-214 - Information Exposure Through Process Environment
- SANS Top 25 - Insecure Interaction Between Components

Deprecated

This rule is deprecated, and will eventually be removed.

Available In:

sonarcloud sonarqube