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

HTTP response headers should not be vulnerable to injection attacks

Analyze your code

Vulnerability Minor injection

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Applications constructing HTTP response headers based on tainted data could allow attackers to change security sensitive headers like Cross-Origin Resource Sharing headers.

Web application frameworks and servers might also allow attackers to inject new line characters in headers to craft malformed HTTP response. In this case the application would be vulnerable to a larger range of attacks like HTTP Response Splitting/Smuggling. Most of the time this type of attack is mitigated by default modern web application frameworks but there might be rare cases where older versions are still vulnerable.

As a best practice, applications that use user-provided data to construct the response header should always validate the data first. Validation should be based on a whitelist.

Noncompliant Code Example

```
$value = $_GET["value"];
header("X-Header: $value"); // Noncompliant
```

Compliant Solution

```
$value = $_GET["value"];
if (ctype_alnum($value)) {
  header("X-Header: $value"); // Compliant
} else {
  // Error
}
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Attack Category - HTTP Response Splitting
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-113 - Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
- SANS Top 25 - Insecure Interaction Between Components

Deprecated

This rule is deprecated; use {rule:php:S5122}, {rule:php:S5146}, {rule:php:S6287} instead.

Available In:

Code Smell

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

sonarcloud sonarqube Developer Edition