- 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...

Vulnerability

XPath expressions should not be vulnerable to injection attacks

Vulnerability

I/O function calls should not be vulnerable to path injection attacks

Vulnerability

LDAP queries should not be vulnerable to injection attacks

Vulnerability

OS commands should not be vulnerable to command injection attacks

Vulnerability

Class of caught exception should be defined

Bug

Caught Exceptions must derive from Throwable

Bug

Raised Exceptions must derive from Throwable

Bug

"$this" should not be used in a static context

Bug

Hard-coded credentials are security-sensitive

Security Hotspot

Test class names should end with "Test"

Code Smell

Tests should include assertions

Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks

Analyze your code

Vulnerability | Blocker | injection cwe sans-top25 owasp

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Furthermore, when processing an HTTP request, a web server may copy user-provided data into the body of the HTTP response that is sent back to the user. This behavior is called a "reflection". Endpoints reflecting tainted data could allow attackers to inject code that would eventually be executed in the user's browser. This could enable a wide range of serious attacks like accessing/modifying sensitive information or impersonating other users.

Typically, the solution is one of the following:

- Validate user-provided data based on a whitelist and reject input that is not allowed.
- Sanitize user-provided data from any characters that can be used for malicious purposes.
- Encode user-provided data when it is reflected back in the HTTP response. Adjust the encoding to the output context so that, for example, HTML encoding is used for HTML content, HTML attribute encoding is used for attribute values, and JavaScript encoding is used for server-generated JavaScript.

When sanitizing or encoding data, it is recommended to only use libraries specifically designed for security purposes. Also, make sure that the library you are using is being actively maintained and is kept up-to-date with the latest discovered vulnerabilities.

Noncompliant Code Example

```
$name = $_GET["name"];
echo "Welcome $name"; // Noncompliant
```

Compliant Solution

```
$name = $_GET["name"];
$safename = htmlspecialchars($name);
echo "Welcome $safename";
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Cheat Sheet - XSS Prevention Cheat Sheet
- OWASP Top 10 2017 Category A7 - Cross-Site Scripting (XSS)
- MITRE, CWE-79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- SANS Top 25 - Insecure Interaction Between Components

Code Smell

TestCases should contain tests

Code Smell

Variable variables should not be used

Code Smell

A new session should be created during user authentication

Vulnerability

Cipher algorithms should be robust

Vulnerability

Encryption algorithms should be used with secure mode and padding

Available In:

sonarcloud sonarqube Developer Edition