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...

Test
Code Smell

Tests should include assertions
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 scheme
Vulnerability

Server hostnames should be verified during SSL/TLS connections
Vulnerability

Server certificates should be verified during SSL/TLS connections
Vulnerability

LDAP connections should be authenticated
Vulnerability

Cryptographic keys should be robust
Vulnerability

Weak SSL/TLS protocols should not be used
Vulnerability

XPath expressions should not be vulnerable to injection attacks

Analyze your code

Vulnerability | Blocker | injection cwe owasp

User-provided data, such as URL parameters, should always be considered untrusted and tainted. Constructing XPath expressions directly from tainted data enables attackers to inject specially crafted values that changes the initial meaning of the expression itself. Successful XPath injection attacks can read sensitive information from XML documents.

Noncompliant Code Example

```
$user = $_GET["user"];
$pass = $_GET["pass"];

$doc = new DOMDocument();
$doc->load("test.xml");
$xpath = new DOMXPath($doc);

$expression = "/users/user[@name='" . $user . "' and @p
$xpath->evaluate($expression); // Noncompliant
```

Compliant Solution

```
$user = $_GET["user"];
$pass = $_GET["pass"];

$doc = new DOMDocument();
$doc->load("test.xml");
$xpath = new DOMXPath($doc);

$user = str_replace("'", "&apos;", $user);
$pass = str_replace("'", "&apos;", $pass);

$expression = "/users/user[@name='" . $user . "' and @p
$xpath->evaluate($expression); // Compliant
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-643 - Improper Neutralization of Data within XPath Expressions

Available In:

sonarcloud | sonarqube Developer Edition

Regular expressions should not be vulnerable to Denial of Service attacks

Vulnerability

Hashes should include an unpredictable salt

Vulnerability

Regular expressions should have valid delimiters

Bug

Regex lookahead assertions should not be contradictory

Bug

Back references in regular expressions should only refer to