sonar RULES

Products

- 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...

HTTP responses should not be vulnerable to session fixation
Vulnerability

Include statements should not be vulnerable to injection attacks
Vulnerability

Dynamic code execution should not be vulnerable to injection attacks
Vulnerability

HTTP request redirections should not be open to forging attacks
Vulnerability

Deserialization should not be vulnerable to injection attacks
Vulnerability

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

Database queries should not be vulnerable to injection attacks
Vulnerability

XML parsers should not be vulnerable to XXE attacks
Vulnerability

A secure password should be used when connecting to a database
Vulnerability

XPath expressions should not be vulnerable to injection attacks
Vulnerability

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

Include statements should not be vulnerable to injection attacks

Analyze your code

Vulnerability | Blocker | injection cwe owasp sans-top25

User-provided data such as URL parameters, POST data payloads or cookies should always be considered untrusted and tainted. Constructing include statements based on data supplied by the user could enable an attacker to control which files are included. If the attacker has the ability to upload files to the system, then arbitrary code could be executed. This could enable a wide range of serious attacks like accessing/modifying sensitive information or gain full system access.

The mitigation strategy should be based on whitelisting of allowed values or casting to safe types.

Noncompliant Code Example

```
$filename = $_GET["filename"];
include $filename . ".php";
```

Compliant Solution

```
$filename = $_GET["filename"];
if (in_array($filename, $whitelist)) {
  include $filename . ".php";
}
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2021 Category A8 - Software and Data Integrity Failures
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-97 - Improper Neutralization of Server-Side Includes (SSI) Within a Web Page
- MITRE, CWE-98 - Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')
- MITRE, CWE-829 - Inclusion of Functionality from Untrusted Control Sphere
- SANS Top 25 - Risky Resource Management

Available In:
sonarcloud sonarqube Developer Edition

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