

- 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 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
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	

Dynamic code execution should not be vulnerable to injection attacks

Analyze your code

Vulnerability

Blocker

injection cwe owasp sans-top25

Applications that execute code dynamically should neutralize any externally-provided values used to construct the code. Failure to do so could allow an attacker to execute arbitrary code. 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

```
$data = $_GET["data"];
eval("echo \"\${data}\";");
```

Compliant Solution

```
$data = $_GET["data"];
if (in_array($data, $whitelist)) {
    eval("echo \"\${data}\";");
}
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-95 - Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
- SANS Top 25 - Risky Resource Management

Available In:

sonarcloud | sonarqube Developer Edition

 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