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...

Code Smell

Single-character alternations in regular expressions should be replaced with character classes

Code Smell

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string

Code Smell

Character classes in regular expressions should not contain the same character twice

Code Smell

Regular expressions should not be too complicated

Code Smell

PHPUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion

Code Smell

Methods should not have identical implementations

Code Smell

Functions should use "return" consistently

Code Smell

Assertion arguments should be passed in the correct order

Code Smell

Ternary operators should not be nested

Code Smell

Reflection should not be used to increase accessibility of classes, methods, or fields

Authorizations should be based on strong decisions

Analyze your code

Vulnerability | Major | cwe owasp

Authorizations granted or not to users to access resources of an application should be based on strong decisions. For instance, checking whether the user is authenticated or not, has the right roles/privileges. It may also depend on the user's location, or the date, time when the user requests access.

Noncompliant Code Example

In a Symfony web application:

- the `vote` method of a VoterInterface type is not compliant when it returns only an affirmative decision (`ACCESS_GRANTED`):

```
class NoncompliantVoterInterface implements VoterInterf
{
    public function vote(TokenInterface $token, $subjec
    {
        return self::ACCESS_GRANTED; // Noncompliant
    }
}
```

- the `voteOnAttribute` method of a Voter type is not compliant when it returns only an affirmative decision (`true`):

```
class NoncompliantVoter extends Voter
{
    protected function supports(string $attribute, $sub
    {
        return true;
    }

    protected function voteOnAttribute(string $attribut
    {
        return true; // Noncompliant
    }
}
```

In a Laravel web application:

- the `define`, `before`, and `after` methods of a Gate are not compliant when they return only an affirmative decision (`true` or `Response::allow()`):

```
class NoncompliantGuard
{
    public function boot()
    {
        Gate::define('xxx', function ($user) {
            return true; // Noncompliant
```

Code Smell

Multiline blocks should be enclosed in curly braces

Code Smell

Parameters should be passed in the correct order

Code Smell

Classes named like "Exception" should extend "Exception" or a subclass

Code Smell

Two branches in a conditional structure should not have exactly the same implementation

```
        });

        Gate::define('xxx', function ($user) {
            return Response::allow(); // Noncompliant
        });
    }
}
```

Compliant Solution

In a Symfony web application:

- the `vote` method of a VoterInterface type should return a negative decision (`ACCESS_DENIED`) or abstain from making a decision (`ACCESS_ABSTAIN`):

```
class CompliantVoterInterface implements VoterInterface
{
    public function vote(TokenInterface $token, $subjec
    {
        if (foo()) {
            return self::ACCESS_GRANTED; // Compliant
        } else if (bar()) {
            return self::ACCESS_ABSTAIN;
        }
        return self::ACCESS_DENIED;
    }
}
```

- the `voteOnAttribute` method of a Voter type should return a negative decision (`false`):

```
class CompliantVoter extends Voter
{
    protected function supports(string $attribute, $sub
    {
        return true;
    }

    protected function voteOnAttribute(string $attribut
    {
        if (foo()) {
            return true; // Compliant
        }
        return false;
    }
}
```

In a Laravel web application:

- the `define`, `before`, and `after` methods of a Gate should return a negative decision (`false` or `Response::deny()`) or abstain from making a decision (`null`):

```
class NoncompliantGuard
{
    public function boot()
    {
        Gate::define('xxx', function ($user) {
            if (foo()) {
                return true; // Compliant
            }
            return false;
        });

        Gate::define('xxx', function ($user) {
            if (foo()) {
                return Response::allow(); // Compliant
            }
            return Response::deny();
        });
    }
}
```

See

- OWASP Top 10 2021 Category A1 - Broken Access Control
- OWASP Top 10 2017 Category A5 - Broken Access Control
- MITRE, CWE-285 - Improper Authorization

Available In:

sonarlint | sonarcloud | sonarqube