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

Constant names should comply with a naming convention
Code Smell

Secret keys and salt values should be robust
Vulnerability

Authorizations should be based on strong decisions
Vulnerability

Server-side requests should not be vulnerable to forging attacks
Vulnerability

The number of arguments passed to a function should match the number of parameters
Bug

Non-empty statements should change control flow or have at least one side-effect
Bug

Variables should be initialized before use
Bug

Replacement strings should reference existing regular expression groups
Bug

Alternation in regular expressions should not contain empty alternatives
Bug

Unicode Grapheme Clusters should be avoided inside regex character classes
Bug

Assertions should not compare an object to itself

Hashes should include an unpredictable salt

Analyze your code

Vulnerability | Critical | cwe sans-top25 owasp

In cryptography, a "salt" is an extra piece of data which is included when hashing a password. This makes `rainbow-table attacks` more difficult. Using a cryptographic hash function without an unpredictable salt increases the likelihood that an attacker could successfully find the hash value in databases of precomputed hashes (called `rainbow-tables`).

This rule raises an issue when a hashing function which has been specifically designed for hashing passwords, such as `PBKDF2`, is used with a non-random, reused or too short salt value. It does not raise an issue on base hashing algorithms such as `sha1` or `md5` as they should not be used to hash passwords.

Recommended Secure Coding Practices

- Use hashing functions generating their own secure salt or generate a secure random value of at least 16 bytes.
- The salt should be unique by user password.

Noncompliant Code Example

```
function createMyAccount() {
  $email = $_GET['email'];
  $name = $_GET['name'];
  $password = $_GET['password'];

  $hash = hash_pbkdf2('sha256', $password, $email, 100000);

  $hash = hash_pbkdf2('sha256', $password, '', 100000); // N

  $hash = hash_pbkdf2('sha256', $password, 'D8VxSmTZt2E2YV45

  $hash = crypt($password); // Noncompliant; salt is not pro
  $hash = crypt($password, ""); // Noncompliant; salt is har

  $options = [
    'cost' => 11,
    'salt' => mcrypt_create_iv(22, MCRYPT_DEV_URANDOM), // N
  ];
  echo password_hash("rasmuslerdorf", PASSWORD_BCRYPT, $opti
}
```

Compliant Solution

```
$salt = openssl_random_pseudo_bytes(16);
$hash = hash_pbkdf2("sha256", $password, $salt, $iterations,
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure

Bug

Regex alternatives should not be redundant

Bug

Alternatives in regular expressions should be grouped when used with anchors

Bug

Array values should not be replaced unconditionally

Bug

- MITRE, CWE-759 - Use of a One-Way Hash without a Salt
- MITRE, CWE-760 - Use of a One-Way Hash with a Predictable Salt
- SANS Top 25 - Porous Defenses

Available In:

sonarlint | sonarcloud | sonarqube