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...

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

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

LDAP queries 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 as untrusted and tainted. Constructing LDAP names or search filters directly from tainted data enables attackers to inject specially crafted values that changes the initial meaning of the name or filter itself. Successful LDAP injections attacks can read, modify or delete sensitive information from the directory service.

Within LDAP names, the special characters `' '`, `'#'`, `'"'`, `'+'`, `','`, `';'`, `'<'`, `'>'`, `'\'` and `null` must be escaped according to RFC 4514, for example by replacing them with the backslash character `'\'` followed by the two hex digits corresponding to the ASCII code of the character to be escaped. Similarly, LDAP search filters must escape a different set of special characters (including but not limited to `'*'`, `'('`, `')'`, `'\'` and `null`) according to RFC 4515.

Noncompliant Code Example

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

$filter = "(&(uid=" . $user . ")(userPassword=" . $pass

$ds = ...
$basedn = "o=My Company, c=US";

$sr = ldap_list($ds, $basedn, $filter); // Noncompliant
```

Compliant Solution

```
function sanitize_ldap_criteria($val) {
  $val = str_replace(['\\', '*', '(', ')'], ['\5c', '\2
  for ($i = 0; $i<strlen($val); $i++) {
    $char = substr($val, $i, 1);
    if (ord($char)<32) {
      $hex = dechex(ord($char));
      if (strlen($hex) == 1) $hex = '0' . $hex;
      $val = str_replace($char, '\\' . $hex, $val);
    }
  }
  return $val;
}

$user = sanitize_ldap_criteria( $_GET["user"] );
$pass = sanitize_ldap_criteria( $_GET["pass"] );

$filter = "(&(uid=" . $user . ")(userPassword=" . $pass
```

Back references in regular expressions should only refer to capturing groups that are matched before the reference

Bug

Regex boundaries should not be used in a way that can never be matched

Bug

Regex patterns following a possessive quantifier should not always fail

Bug

Assertion failure exceptions should not be ignored

Bug

```
$ds = ...
$basedn = "o=My Company, c=US";

$sr = ldap_list($ds, $basedn, $filter);
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- RFC 4514 - LDAP: String Representation of Distinguished Names
- RFC 4515 - LDAP: String Representation of Search Filters
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-90 - Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

Available In:

sonarcloud | sonarqube Developer Edition