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

Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

All rules 632 | Vulnerability 53 | Bug 154 | Security Hotspot 36 | Code Smell 389 | Quick Fix 42

Tags | Search by name...

Methods should not call same-class methods with incompatible "@Transactional" values
Bug

Recursion should not be infinite
Bug

Loops should not be infinite
Bug

Double-checked locking should not be used
Bug

Resources should be closed
Bug

Hard-coded credentials are security-sensitive
Security Hotspot

Methods returns should not be invariant
Code Smell

"ThreadGroup" should not be used
Code Smell

"clone" should not be overridden
Code Smell

Assertions should be complete
Code Smell

Tests should include assertions
Code Smell

Silly bit operations should not be performed
Code Smell

Child class fields should not shadow parent class fields

LDAP queries should not be vulnerable to injection attacks

Analyze your code

Vulnerability | Blocker | injection cwe owasp cert

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

```
public boolean authenticate(javax.servlet.http.HttpServletRe
  String user = request.getParameter("user");
  String pass = request.getParameter("pass");

  String filter = "(&(uid=" + user + ")(userPassword=" + pas

  // If the special value "*)(uid=*))(|(uid=*" is passed as
  // Indeed, if it is passed as a user, the filter becomes:
  // (&(uid=*)(uid=*))(|(uid=*)(userPassword=...))
  // as uid=* match all users, it is equivalent to:
  // (|(uid=*)(userPassword=...))
  // again, as uid=* match all users, the filter becomes use

  NamingEnumeration<SearchResult> results = ctx.search("ou=s
  return results.hasMore();
}
```

Compliant Solution

```
public boolean authenticate(javax.servlet.http.HttpServletRe
  String user = request.getParameter("user");
  String pass = request.getParameter("pass");

  String filter = "(&(uid={0})(userPassword={1}))"; // Safe

  NamingEnumeration<SearchResult> results = ctx.search("ou=s
  return results.hasMore();
}
```

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-90 - Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
- CERT, IDS54-J. - Prevent LDAP injection

Available In:

sonarcloud sonarqube Developer Edition

Code Smell

JUnit test cases should call super methods

Code Smell

TestCases should contain tests

Code Smell

Short-circuit logic should be used in boolean contexts

Code Smell

Methods and field names should not be the same or differ only by capitalization