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...

"entrySet()" should be iterated when both the key and value are needed
Code Smell

"DateUtils.truncate" from Apache Commons Lang library should not be used
Code Smell

Multiline blocks should be enclosed in curly braces
Code Smell

"readObject" should not be "synchronized"
Code Smell

"Preconditions" and logging arguments should not require evaluation
Code Smell

Boolean expressions should not be gratuitous
Code Smell

"Lock" objects should not be "synchronized"
Code Smell

Classes with only "static" methods should not be instantiated
Code Smell

"Threads" should not be used where "Runnables" are expected
Code Smell

Inner class calls to super class methods should be unambiguous
Code Smell

Unused type parameters should be removed
Code Smell

"ThreadLocal" variables should be cleaned up when no longer used

Analyze your code

Bug Major Quick Fix leak performance

`ThreadLocal` variables are supposed to be garbage collected once the holding thread is no longer alive. Memory leaks can occur when holding threads are re-used which is the case on application servers using pool of threads.

To avoid such problems, it is recommended to always clean up `ThreadLocal` variables using the `remove()` method to remove the current thread's value for the `ThreadLocal` variable.

In addition, calling `set(null)` to remove the value might keep the reference to `this` pointer in the map, which can cause memory leak in some scenarios. Using `remove` is safer to avoid this issue.

Noncompliant Code Example

```
public class ThreadLocalUserSession implements UserSession {

  private static final ThreadLocal<UserSession> DELEGATE = n

  public UserSession get() {
    UserSession session = DELEGATE.get();
    if (session != null) {
      return session;
    }
    throw new UnauthorizedException("User is not authenticat
  }

  public void set(UserSession session) {
    DELEGATE.set(session);
  }

   public void incorrectCleanup() {
     DELEGATE.set(null); // Noncompliant
   }

  // some other methods without a call to DELEGATE.remove()
}
```

Compliant Solution

```
public class ThreadLocalUserSession implements UserSession {

  private static final ThreadLocal<UserSession> DELEGATE = n

  public UserSession get() {
    UserSession session = DELEGATE.get();
    if (session != null) {
      return session;
    }
    throw new UnauthorizedException("User is not authenticat
  }
```

Parameters should be passed in the correct order

Code Smell

"ResultSet.isLast()" should not be used

Code Smell

"static" members should be accessed statically

Code Smell

Silly math should not be performed

Code Smell

```
  public void set(UserSession session) {
    DELEGATE.set(session);
  }

  public void unload() {
    DELEGATE.remove(); // Compliant
  }

  // ...
}
```

Exceptions

Rule will not detect non-private `ThreadLocal` variables, because `remove()` can be called from another class.

See

- Understanding Memory Leaks in Java

Available In:

sonarlint sonarcloud sonarqube