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...

Code Smell

Lambdas containing only one statement should not nest this statement in a block

Code Smell

"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used

Code Smell

Local variables should not be declared and then immediately returned or thrown

Code Smell

Unused local variables should be removed

Code Smell

Private fields only used as local variables in methods should become local variables

Code Smell

"public static" fields should be constant

Code Smell

Loops should not contain more than a single "break" or "continue" statement

Code Smell

Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"

Code Smell

"switch" statements should have at least 3 "case" clauses

Code Smell

A "while" loop should be used instead of a "for" loop

Code Smell

Setting loose POSIX file permissions is security-sensitive

Analyze your code

Security Hotspot Major cwe cert sans-top25 owasp

In Unix, "`others`" class refers to all users except the owner of the file and the members of the group assigned to this file.

Granting permissions to this group can lead to unintended access to files.

Ask Yourself Whether

- The application is designed to be run on a multi-user environment.
- Corresponding files and directories may contain confidential information.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

The most restrictive possible permissions should be assigned to files and directories.

Sensitive Code Example

```
public void setPermissions(String filePath) {
    Set<PosixFilePermission> perms = new HashSet<PosixFi
    // user permission
    perms.add(PosixFilePermission.OWNER_READ);
    perms.add(PosixFilePermission.OWNER_WRITE);
    perms.add(PosixFilePermission.OWNER_EXECUTE);
    // group permissions
    perms.add(PosixFilePermission.GROUP_READ);
    perms.add(PosixFilePermission.GROUP_EXECUTE);
    // others permissions
    perms.add(PosixFilePermission.OTHERS_READ); // Sensi
    perms.add(PosixFilePermission.OTHERS_WRITE); // Sens
    perms.add(PosixFilePermission.OTHERS_EXECUTE); // Se

    Files.setPosixFilePermissions(Paths.get(filePath), p
}
```

```
public void setPermissionsUsingRuntimeExec(String filePa
    Runtime.getRuntime().exec("chmod 777 file.json"); //
}
```

```
public void setOthersPermissionsHardCoded(String filePat
    Files.setPosixFilePermissions(Paths.get(filePath), P
}
```

Compliant Solution

On operating systems that implement POSIX standard. This will throw a `UnsupportedOperationException` on Windows.

The default unnamed package should not be used

Code Smell

"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method

Code Smell

Package names should comply with a naming convention

Code Smell

Nested code blocks should not be used

```
    public void setPermissionsSafe(String filePath) throws I
        Set<PosixFilePermission> perms = new HashSet<PosixFi
        // user permission
        perms.add(PosixFilePermission.OWNER_READ);
        perms.add(PosixFilePermission.OWNER_WRITE);
        perms.add(PosixFilePermission.OWNER_EXECUTE);
        // group permissions
        perms.add(PosixFilePermission.GROUP_READ);
        perms.add(PosixFilePermission.GROUP_EXECUTE);
        // others permissions removed
        perms.remove(PosixFilePermission.OTHERS_READ); // Co
        perms.remove(PosixFilePermission.OTHERS_WRITE); // C
        perms.remove(PosixFilePermission.OTHERS_EXECUTE); //

        Files.setPosixFilePermissions(Paths.get(filePath), p
    }
```

See

- OWASP Top 10 2021 Category A1 - Broken Access Control
- OWASP Top 10 2021 Category A4 - Insecure Design
- OWASP Top 10 2017 Category A5 - Broken Access Control
- OWASP File Permission
- MITRE, CWE-732 - Incorrect Permission Assignment for Critical Resource
- MITRE, CWE-266 - Incorrect Privilege Assignment
- CERT, FIO01-J. - Create files with appropriate access permissions
- CERT, FIO06-C. - Create files with appropriate access permissions
- SANS Top 25 - Porous Defenses

Available In:

sonarcloud sonarqube