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

Mutable fields should not be "public static"
Code Smell

The diamond operator ("<>") should be used
Code Smell

"finalize" should not set fields to "null"
Code Smell

Subclasses that add fields should override "equals"
Code Smell

Catches should be combined
Code Smell

Methods of "Random" that return floating point values should not be used in random integer generation
Code Smell

Parsing should be used to convert "Strings" to primitives
Code Smell

Classes should not be empty
Code Smell

Fields in non-serializable classes should not be "transient"
Code Smell

Boolean checks should not be inverted
Code Smell

Redundant casts should not be used
Code Smell

"@Deprecated" code should not be used

Constructing arguments of system commands from user input is security-sensitive

Analyze your code

Security Hotspot | Major | injection cwe owasp sans-top25

Constructing arguments of system commands from user input is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2016-9920
- CVE-2021-29472

Arguments of system commands are processed by the executed program. The arguments are usually used to configure and influence the behavior of the programs. Control over a single argument might be enough for an attacker to trigger dangerous features like executing arbitrary commands or writing files into specific directories.

Ask Yourself Whether

- Malicious arguments can result in undesired behavior in the executed command.
- Passing user input to a system command is not necessary.

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

Recommended Secure Coding Practices

- Avoid constructing system commands from user input when possible.
- Ensure that no risky arguments can be injected for the given program, e.g., type-cast the argument to an integer.
- Use a more secure interface to communicate with other programs, e.g., the standard input stream (stdin).

Sensitive Code Example

Arguments like `-delete` or `-exec` for the `find` command can alter the expected behavior and result in vulnerabilities:

```
String input = request.getParameter("input");
String cmd[] =  new String[] { "/usr/bin/find", input };
Runtime.getRuntime().exec(cmd); // Sensitive
```

Compliant Solution

Use an allow-list to restrict the arguments to trusted values:

```
String input = request.getParameter("input");
if (allowed.contains(input)) {
  String cmd[] =  new String[] { "/usr/bin/find", input };
  Runtime.getRuntime().exec(cmd);
}
```

See

Code Smell

"toString()" should never be called on a String object

Code Smell

Annotation repetitions should not be wrapped

Code Smell

Multiple variables should not be declared on the same line

Code Smell

Strings should not be concatenated using '+' in a loop

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-88 - Argument Injection or Modification
- SANS Top 25 - Insecure Interaction Between Components
- CVE-2021-29472 - PHP Supply Chain Attack on Composer

Available In:

sonarcloud | sonarqube Developer Edition