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...

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
Code Smell

JUnit test cases should call super methods

OS commands should not be vulnerable to command injection attacks

Analyze your code

Vulnerability Blocker injection cwe owasp sans-top25

Applications that allow execution of operating system commands from user-controlled data should control the command to execute, otherwise an attacker can inject arbitrary commands that will compromise the underlying operating system.

The mitigation strategy can be based on a list of authorized and safe commands to execute and when a shell is spawned to sanitize shell meta-characters. Keep in mind that when a single argument to the command is user-controlled and shell-metachars are sanitized, it can still lead to vulnerabilities if the attacker can inject a dangerous option supported by the command, such as `-exec` available with find, in that case, mark end of option processing on the command line using `--` (double-dash) or restrict options to only trusted values.

Noncompliant Code Example

```
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;

public void runUnsafe(HttpServletRequest request) throws IOE
  String cmd = request.getParameter("command");
  String arg = request.getParameter("arg");

  Runtime.getRuntime().exec(cmd+" "+arg); // Noncompliant
}
```

Compliant Solution

Implement an allow-list of authorized commands to execute:

- each time the command to execute is user-controlled:

```
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;

public void runUnsafe(HttpServletRequest request) throws IOE
  String cmd = request.getParameter("command");
  String arg = request.getParameter("arg");

  if(cmd.equals("/usr/bin/ls") || cmd.equals("/usr/bin/cat")
  {
       // only ls or cat command are authorized
       String cmdarray[] =  new String[] { cmd, arg };
       Runtime.getRuntime().exec(cmdarray); // Compliant
  }
}
```

- or globally with the creation of a SecurityManager overriding checkExec() method:

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

Code Smell

Switch cases should end with an unconditional "break" statement

```
class MySecurityManager extends SecurityManager {
  MySecurityManager() {
    super();
  }

  public void checkExec(String cmd) {
    if(!(cmd.equals("/usr/bin/ls") || cmd.equals("/usr/bin/c
      throw new SecurityException("Unauthorized command: "+c
    }
  }
}
```

```
MySecurityManager sm = new MySecurityManager();
System.setSecurityManager(sm);
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP OS Command Injection Defense Cheat Sheet
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-78 - Improper Neutralization of Special Elements used in an OS Command
- SANS Top 25 - Insecure Interaction Between Components

Available In:

sonarcloud sonarqube Developer Edition