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...

Abstract class names should comply with a naming convention
Code Smell

Strings literals should be placed on the left side when checking for equality
Code Smell

Files should contain an empty newline at the end
Code Smell

Source code should be indented consistently
Code Smell

A close curly brace should be located at the beginning of a line
Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines
Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line
Code Smell

An open curly brace should be located at the beginning of a line
Code Smell

An open curly brace should be located at the end of a line
Code Smell

Tabulation characters should not be used
Code Smell

Functions should not be defined with a variable number of arguments
Code Smell

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive

Analyze your code

Security Hotspot Minor cwe spring owasp sans-top25

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2018-0269
- CVE-2017-14460

Same origin policy in browsers prevents, by default and for security-reasons, a javascript frontend to perform a cross-origin HTTP request to a resource that has a different origin (domain, protocol, or port) from its own. The requested target can append additional HTTP headers in response, called CORS, that act like directives for the browser and change the access control policy / relax the same origin policy.

Ask Yourself Whether

- You don't trust the origin specified, example: `Access-Control-Allow-Origin: untrustedwebsite.com`.
- Access control policy is entirely disabled: `Access-Control-Allow-Origin: *`
- Your access control policy is dynamically defined by a user-controlled input like `origin` header.

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

Recommended Secure Coding Practices

- The `Access-Control-Allow-Origin` header should be set only for a trusted origin and for specific resources.
- Allow only selected, trusted domains in the `Access-Control-Allow-Origin` header. Prefer whitelisting domains over blacklisting or allowing any domain (do not use * wildcard nor blindly return the `Origin` header content without any checks).

Sensitive Code Example

Java servlet framework:

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResp
    resp.setHeader("Content-Type", "text/plain; charset=utf-
    resp.setHeader("Access-Control-Allow-Origin", "*"); // S
    resp.setHeader("Access-Control-Allow-Credentials", "true
    resp.setHeader("Access-Control-Allow-Methods", "GET");
    resp.getWriter().write("response");
}
```

Spring MVC framework:

CrossOrigin

```
@CrossOrigin // Sensitive
@RequestMapping("")
```

Local-Variable Type Inference should be used

Code Smell

Migrate your tests from JUnit4 to the new JUnit5 annotations

Code Smell

Track uses of disallowed classes

Code Smell

Track uses of "@SuppressWarnings" annotations

Code Smell

```
public class TestController {
    public String home(ModelMap model) {
        model.addAttribute("message", "ok ");
        return "view";
    }
}
```

cors.CorsConfiguration

```
CorsConfiguration config = new CorsConfiguration();
config.addAllowedOrigin("*"); // Sensitive
config.applyPermitDefaultValues(); // Sensitive
```

servlet.config.annotation.CorsConfiguration

```
class Insecure implements WebMvcConfigurer {
  @Override
  public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**")
      .allowedOrigins("*"); // Sensitive
  }
}
```

Compliant Solution

Java Servlet framework:

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResp
    resp.setHeader("Content-Type", "text/plain; charset=utf-
    resp.setHeader("Access-Control-Allow-Origin", "trustedwe
    resp.setHeader("Access-Control-Allow-Credentials", "true
    resp.setHeader("Access-Control-Allow-Methods", "GET");
    resp.getWriter().write("response");
}
```

Spring MVC framework:

CrossOrigin

```
@CrossOrigin("trustedwebsite.com") // Compliant
@RequestMapping("")
public class TestController {
    public String home(ModelMap model) {
        model.addAttribute("message", "ok ");
        return "view";
    }
}
```

cors.CorsConfiguration

```
CorsConfiguration config = new CorsConfiguration();
config.addAllowedOrigin("http://domain2.com"); // Compliant
```

servlet.config.annotation.CorsConfiguration

```
class Safe implements WebMvcConfigurer {
  @Override
  public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**")
      .allowedOrigins("safe.com"); // Compliant
  }
}
```

See

- OWASP Top 10 2021 Category A5 - Security Misconfiguration
- OWASP Top 10 2021 Category A7 - Identification and Authentication Failures
- developer.mozilla.org - CORS
- developer.mozilla.org - Same origin policy
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- OWASP HTML5 Security Cheat Sheet - Cross Origin Resource Sharing
- MITRE, CWE-346 - Origin Validation Error
- MITRE, CWE-942 - Overly Permissive Cross-domain Whitelist

- SANS Top 25 - Porous Defenses

Available In:

sonarcloud | sonarqube