














XML static code analysis

Unique rules to find Bugs and Code Smells in your XML code

All rules 36 Vulnerability 6 Bug 5 Security Hotspot 9 Code Smell 16

Tags

Search by name...

| | |
|---|--|
| Having a permissive Cross-Origin Resource Sharing policy is security-sensitive |  Security Hotspot |
| Delivering code in production with debug features activated is security-sensitive |  Security Hotspot |
| Creating cookies without the "HttpOnly" flag is security-sensitive |  Security Hotspot |
| Deprecated "\${pom}" properties should not be used |  Code Smell |
| Track uses of "TODO" tags |  Code Smell |
| EJB interceptor exclusions should be declared as annotations |  Code Smell |
| Track uses of disallowed dependencies |  Code Smell |
| Newlines should follow each element |  Code Smell |
| XML parser failure |  Code Smell |
| Track breaches of an XPath rule |  Code Smell |
| Lines should not be too long |  Code Smell |
| pom elements should be in the recommended order |  Code Smell |
| Artifact ids should follow a naming convention |  Code Smell |

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

Analyze your code

Security Hotspot Minor cwe 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

```
<!-- Tomcat 7+ Cors Filter -->
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
  <init-param>
    <param-name>cors.allowed.origins</param-name>
    <param-value>*</param-value> <!-- Sensitive -->
  </init-param>
</filter>
```

Compliant Solution

```
<!-- Tomcat 7+ Cors Filter -->
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
  <init-param>
    <param-name>cors.allowed.origins</param-name>
    <param-value>https://trusted1.org,https://trusted2.org</param-value> <!-- Compliant -->
  </init-param>
</filter>
```

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 