Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

Maps with keys that are enum values should be replaced with EnumMap
Code Smell

Lambdas should be replaced with method references
Code Smell

Parentheses should be removed from a single lambda input parameter when its type is inferred
Code Smell

Abstract classes without fields should be converted to interfaces
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

Disabling auto-escaping in template engines is security-sensitive

Analyze your code

Security Hotspot   Major   cwe owasp

To reduce the risk of cross-site scripting attacks, templating systems, such as `Twig`, `Django`, `Smarty`, `Groovy's template engine`, allow configuration of automatic variable escaping before rendering templates. When escape occurs, characters that make sense to the browser (eg: <a>) will be transformed/replaced with escaped/sanitized values (eg: & lt;a& gt;).

Auto-escaping is not a magic feature to annihilate all cross-site scripting attacks, it depends on the strategy applied and the context, for example a "html auto-escaping" strategy (which only transforms html characters into html entities) will not be relevant when variables are used in a html attribute because '`:`' character is not escaped and thus an attack as below is possible:

```
<a href="{{ myLink }}">link</a> // myLink = javascript:alert
<a href="javascript:alert(document.cookie)">link</a> // JS i
```

Ask Yourself Whether

- Templates are used to render web content and
  - dynamic variables in templates come from untrusted locations or are user-controlled inputs
  - there is no local mechanism in place to sanitize or validate the inputs.

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

Recommended Secure Coding Practices

Enable auto-escaping by default and continue to review the use of inputs in order to be sure that the chosen auto-escaping strategy is the right one.

Sensitive Code Example

With JMustache by samskivert:

```
Mustache.compiler().escapeHTML(false).compile(template).exec
Mustache.compiler().withEscaper(Escapers.NONE).compile(templ
```

With Freemarker:

```
freemarker.template.Configuration configuration = new freema
configuration.setAutoEscapingPolicy(DISABLE_AUTO_ESCAPING_PO
```

Compliant Solution

With JMustache by samskivert:

```
Mustache.compiler().compile(template).execute(context); // C
Mustache.compiler().escapeHTML(true).compile(template).execu
```

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

With Freemarker. See "setAutoEscapingPolicy" documentation for more details.

```
freemarker.template.Configuration configuration = new freema
configuration.setAutoEscapingPolicy(ENABLE_IF_DEFAULT_AUTO_E
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Cheat Sheet - XSS Prevention Cheat Sheet
- OWASP Top 10 2017 Category A7 - Cross-Site Scripting (XSS)
- MITRE, CWE-79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Available In:

sonarcloud sonarqube