




 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... 🔍

is security-sensitive

Security Hotspot

Formatting SQL queries is security-sensitive

Security Hotspot

Deprecated annotations should include explanations

Code Smell

Restricted Identifiers should not be used as Identifiers

Code Smell

Redundant constructors/methods should be avoided in records

Code Smell

Records should be used instead of ordinary classes when representing immutable data structure

Code Smell

"Stream.toList()" method should be used instead of "collectors" when unmodifiable list needed

Code Smell

Operator "instanceof" should be used instead of "A.class.isInstance()"

Code Smell

String multiline concatenation should be replaced with Text Blocks

Code Smell

Single-character alternations in regular expressions should be replaced with character classes

Code Smell

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string

Code Smell

JSON operations should not be vulnerable to injection attacks

Analyze your code

Vulnerability

Major

injection cwe owasp

User-provided data, such as URL parameters, POST data payloads, or cookies, are tainted with user-controlled inputs. Therefore, they should always be considered untrusted.

JSON injection occurs when user-controlled input is embedded into JSON syntax.

In such a scenario, an attacker can alter the semantics of the underlying JSON object and insert additional properties, modify existing properties or change the JSON object in some other way that potentially affects the business logic of the program processing that JSON object.

An attacker may exploit such a vulnerability in a variety of ways, depending on the business logic. In the worst case, it may even be possible for the attacker to execute arbitrary code.

Noncompliant Code Example

```
protected void doGet(HttpServletRequest req, HttpServletResponse
String json = "{\key\":\"+req.getParameter("value")+"\
FileOutputStream fos = new FileOutputStream("output.json")
fos.write(json.getBytes(Charset.forName("UTF-8"))); // Noncompliant
}
```

[google/gson](#)

```
protected void doGet(HttpServletRequest req, HttpServletResponse
Gson gson = new Gson();
String json = "{\key\":\"+req.getParameter("value")+"\
Object object = gson.fromJson(json, Object.class); // Noncompliant
}
```

[FasterXML/jackson](#)

```
protected void doGet(HttpServletRequest req, HttpServletResponse
ObjectMapper mapper = new ObjectMapper();
JsonFactory jsonFactory = mapper.getFactory();
JsonGenerator generator = jsonFactory.createGenerator(Syst

String json = "{\key\":\"+req.getParameter("value")+"\
generator.writeRaw(json); // Noncompliant
generator.close();
}
```

[stleary/JSON-java](#) (org.json)

```
protected void doGet(HttpServletRequest req, HttpServletResponse
String json = "{\key\":\"+req.getParameter("value")+"\
JSONObject jo = new JSONObject(json); // Noncompliant
}
```

https://rules.sonarsource.com/java/RSPEC-6398

1/2

Constructors of an "abstract" class should not be declared "public"

 Code Smell

Similar tests should be grouped in a single Parameterized test

 Code Smell

Tests should be stable

 Code Smell

Test methods should not contain too many assertions

 Code Smell

Compliant Solution

```
protected void doGet(HttpServletRequest req, HttpServletResponse
String value = req.getParameter("value");
if(value.matches("[A-Za-z0-9_]+$")) {
    String json = "{\"key\":\""+value+"\"}";
    FileOutputStream fos = new FileOutputStream("output.json");
    fos.write(json.getBytes(Charset.forName("UTF-8")));
}
}
```

[google/gson](#)

```
protected void doGet(HttpServletRequest req, HttpServletResponse
Gson gson = new Gson();
JsonObject json = new JsonObject();
json.addProperty("key", req.getParameter("value"));
Object object = gson.fromJson(json, Object.class);
}
```

[FasterXML/jackson](#)

```
protected void doGet(HttpServletRequest req, HttpServletResponse
ObjectMapper mapper = new ObjectMapper();
JsonFactory jsonFactory = mapper.getFactory();
JsonGenerator generator = jsonFactory.createGenerator(Syst

generator.writeStartObject();
enerator.writeFieldName("key");
generator.writeString(req.getParameter("value"));
generator.close();
}
```

[stleary/JSON-java](#) (org.json)

```
protected void doGet(HttpServletRequest req, HttpServletResponse
JSONObject jo = new JSONObject();
jo.append("key", req.getParameter("value"));
}
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-20](#) - Improper Input Validation
- [MITRE, CWE-76](#) - Improper Neutralization of Equivalent Special Elements

Available In:

sonarcloud  **sonarqube**  Developer Edition