




 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 rules632

Vulnerability53

Bug154


Security Hotspot36

Code Smell389


Quick Fix42


Tags ▾

Search by name... 🔍


 Vulnerability

LDAP queries should not be vulnerable to injection attacks


 Vulnerability

 Bug


"@SpringBootApplication" and "@ComponentScan" should not be used in the default package

 Bug


"@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects

 Bug


"wait" should not be called when multiple locks are held

 Bug


"PreparedStatement" and "ResultSet" methods should be called with valid indices

 Bug


Files opened in append mode should not be used with ObjectOutputStream

 Bug

"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held

 Bug




Printf-style format strings should not lead to unexpected behavior at runtime

 Bug

Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances

Deserialization should not be vulnerable to injection attacks

Analyze your code

 Vulnerability  Blocker  injection cwe sans-top25 owasp

User-provided data such as URL parameters, POST data payloads or cookies should always be considered untrusted and tainted. Deserialization based on data supplied by the user could result in two types of attacks:

- Remote code execution attacks, where the structure of the serialized data is changed to modify the behavior of the object being unserialized.
- Parameter tampering attacks, where data is modified to escalate privileges or change for example quantity or price of products.

The best way to protect against deserialization attacks is probably to challenge the use of the deserialization mechanism in the application. They are cases where the use of deserialization mechanism was not justified and created breaches (CVE-2017-9785).

If the use of deserialization mechanisms is valid in your context, the problem could be mitigated in any of the following ways:

- Instead of using a native data interchange format, use a safe, standard format such as untyped JSON or structured data approaches such as Google Protocol Buffers.
- To ensure integrity is not compromised, add a digital signature (HMAC) to the serialized data that is validated before deserialization (this is only valid if the client doesn't need to modify the serialized data)
- As a last resort, restrict deserialization to be possible only to specific, whitelisted classes.

This rule supports: Java native serialization, XMLEncoder and Kryo





Noncompliant Code Example

```
public class RequestProcessor {
    protected void processRequest(HttpServletRequest request) {
        ServletInputStream sis = request.getInputStream();
        ObjectInputStream ois = new ObjectInputStream(sis);
        Object obj = ois.readObject(); // Noncompliant
    }
}
```

Compliant Solution

```
public class SecureObjectInputStream extends ObjectInputStream {
    // Constructor here

    @Override
    protected Class<?> resolveClass(ObjectStreamClass osc) throws IOException {
        // Only deserialize instances of AllowedClass
        if (!osc.getName().equals(AllowedClass.class.getName())) {
            throw new InvalidClassException("Unauthorized deserialization");
        }
        return super.resolveClass(osc);
    }
}
```

Methods should not call same-class methods with incompatible "@Transactional" values  Bug
Recursion should not be infinite  Bug
Loops should not be infinite  Bug
Double-checked locking should not be used  Bug

```
}  
  
public class RequestProcessor {  
    protected void processRequest(HttpServletRequest request)  
    {  
        ServletInputStream sis = request.getInputStream();  
        SecureObjectInputStream sois = new SecureObjectInputStre  
        Object obj = sois.readObject();  
    }  
}
```

See

- [OWASP Top 10 2021 Category A8](#) - Software and Data Integrity Failures
- [OWASP Top 10 2017 Category A8](#) - Insecure Deserialization
- [MITRE, CWE-20](#) - Improper Input Validation
- [MITRE, CWE-502](#) - Deserialization of Untrusted Data
- [SANS Top 25](#) - Risky Resource Management

Available In:
  