

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

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...

Generating resource consumption is security-sensitive

Security Hotspot

Configuring loggers is security-sensitive

Security Hotspot

Using weak hashing algorithms is security-sensitive

Security Hotspot

Using unsafe Jackson deserialization configuration is security-sensitive

Security Hotspot

Setting JavaBean properties is security-sensitive

Security Hotspot

Disabling CSRF protections is security-sensitive

Security Hotspot

Using non-standard cryptographic algorithms is security-sensitive

Security Hotspot

Using pseudorandom number generators (PRNGs) is security-sensitive

Security Hotspot

Mocking all non-private methods of a class should be avoided

Code Smell

Empty lines should not be tested with regex MULTILINE flag

Code Smell

Methods setUp() and tearDown() should be correctly annotated starting with JUnit4

Code Smell

Class members annotated with "@RequestMapping" should not be used as arguments of "@RequestMapping" methods

Security Hotspot

Persistent entities should not be used as arguments of "@RequestMapping" methods

Analyze your code

Vulnerability

Critical

cwe spring owasp

On one side, Spring MVC automatically bind request parameters to beans declared as arguments of methods annotated with @RequestMapping. Because of this automatic binding feature, it's possible to feed some unexpected fields on the arguments of the @RequestMapping annotated methods.

On the other end, persistent objects (@Entity or @Document) are linked to the underlying database and updated automatically by a persistence framework, such as Hibernate, JPA or Spring Data MongoDB.

These two facts combined together can lead to malicious attack: if a persistent object is used as an argument of a method annotated with @RequestMapping, it's possible from a specially crafted user input, to change the content of unexpected fields into the database.

For this reason, using @Entity or @Document objects as arguments of methods annotated with @RequestMapping should be avoided.

In addition to @RequestMapping, this rule also considers the annotations introduced in Spring Framework 4.3: @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping.

Noncompliant Code Example

```
import javax.persistence.Entity;

@Entity
public class Wish {
    Long productId;
    Long quantity;
    Client client;
}

@Entity
public class Client {
    String clientId;
    String name;
    String password;
}

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class WishListController {

    @PostMapping(path = "/saveForLater")
    public String saveForLater(Wish wish) {
        session.save(wish);
    }

    @RequestMapping(path = "/saveForLater", method = RequestMethod.POST)
    public String saveForLater(@RequestBody Wish wish) {
        session.save(wish);
    }
}
```

https://rules.sonarsource.com/java/RSPEC-4684

1/2

"@VisibleForTesting" should not be accessed from production code

Code Smell

"String#replace" should be preferred to "String#replaceAll"

Code Smell

Derived exceptions should not hide their parents' catch blocks

Code Smell

String offset-based methods should be preferred for finding substrings from offsets

Code Smell

```
public String saveForLater(Wish wish) {  
    session.save(wish);  
}
```

Compliant Solution

```
public class WishDTO {  
    Long productId;  
    Long quantity;  
    Long clientId;  
}  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
  
@Controller  
public class PurchaseOrderController {  
  
    @PostMapping(path = "/saveForLater")  
    public String saveForLater(WishDTO wish) {  
        Wish persistentWish = new Wish();  
        // do the mapping between "wish" and "persistentWish"  
        [...]  
        session.save(persistentWish);  
    }  
  
    @RequestMapping(path = "/saveForLater", method = RequestMethod.  
    public String saveForLater(WishDTO wish) {  
        Wish persistentWish = new Wish();  
        // do the mapping between "wish" and "persistentWish"  
        [...]  
        session.save(persistentWish);  
    }  
}
```

Exceptions

No issue is reported when the parameter is annotated with @PathVariable from Spring Framework, since the lookup will be done via id, the object cannot be forged on client side.

See

- OWASP Top 10 2021 Category A8 - Software and Data Integrity Failures
- OWASP Top 10 2017 Category A5 - Broken Access Control
- MITRE, CWE-915 - Improperly Controlled Modification of Dynamically-Determined Object Attributes
- Two Security Vulnerabilities in the Spring Framework's MVC by Ryan Berg and Dinis Cruz

Available In:

sonarlint | sonarcloud | sonarqube