




 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... 🔍

toString() should never be called on a String object

Code Smell

Annotation repetitions should not be wrapped

Code Smell

Multiple variables should not be declared on the same line

Code Smell

Strings should not be concatenated using '+' in a loop

Code Smell

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

Allowing user enumeration is security-sensitive

Analyze your code

Security HotspotMajor?cwe spring owasp

User enumeration refers to the ability to guess existing usernames in a web application database. This can happen, for example, when using "sign-in/sign-on/forgot password" functionalities of a website.

When a user tries to "sign-in" to a website with an incorrect username/login, the web application should not disclose that the username doesn't exist with a message similar to "this username is incorrect", instead a generic message should be used like "bad credentials", this way it's not possible to guess whether the username or password was incorrect during the authentication.

If a user-management feature discloses information about the existence of a username, attackers can use brute force attacks to retrieve a large amount of valid usernames that will impact the privacy of corresponding users and facilitate other attacks (phishing, password guessing etc ...).

Ask Yourself Whether

- The application discloses that a username exists in its database: most of the time it's possible to avoid this kind of leak except for the "registration/sign-on" part of a website because in this case the user must choose a valid username (not already taken by another user).
- There is no rate limiting and CAPTCHA protection in place for requests involving a username.

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

Recommended Secure Coding Practices

When a user performs a request involving a username, it should not be possible to spot differences between a valid and incorrect username:

- Error messages should be generic and not disclose if the username is valid or not.
- The response time must be similar for a valid username or not.
- CAPTCHA and other rate limiting solutions should be implemented.

Sensitive Code Example

In a Spring-security web application the username leaks when:





- The string used as argument of `loadUserByUsername` method is used in an exception message:

```
public String authenticate(String username, String password)
// ....
MyUserDetailsService s1 = new MyUserDetailsService();
MyUserPrincipal u1 = s1.loadUserByUsername(username);

if(u1 == null) {
    throw new BadCredentialsException(username+" doesn't exi
}
// ....
}
```

https://rules.sonarsource.com/java/RSPEC-5804

1/2

 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

- [UsernameNotFoundException](#) is thrown (except when it is in the loadUserByUsername method):

```
public String authenticate(String username, String password)
// ....
if(user == null) {
    throw new UsernameNotFoundException("user not found");
}
// ....
}
```

- [HideUserNotFoundExceptions](#) is set to false:

```
DaoAuthenticationProvider daoauth = new DaoAuthenticationPro
daoauth.setUserDetailsService(new MyUserDetailsService());
daoauth.setPasswordEncoder(new BCryptPasswordEncoder());
daoauth.setHideUserNotFoundExceptions(false); // Sensitive
builder.authenticationProvider(daoauth);
```

#### Compliant Solution

In a Spring-security web application:

- the same message should be used regardless of whether it is the wrong user or password:

```
public String authenticate(String username, String password)
    Details user = null;
    try {
        user = loadUserByUsername(username);
    } catch (UsernameNotFoundException | DataAccessException e) {
        // Hide this exception reason to not disclose that the u
    }
    if (user == null || !user.isPasswordCorrect(password)) {
        // User should not be able to guess if the bad credenti
        throw new BadCredentialsException("Bad credentials");
    }
}
```

- [HideUserNotFoundExceptions](#) should be set to true:

```
DaoAuthenticationProvider daoauth = new DaoAuthenticationPro
daoauth.setUserDetailsService(new MyUserDetailsService());
daoauth.setPasswordEncoder(new BCryptPasswordEncoder());
daoauth.setHideUserNotFoundExceptions(true); // Compliant
builder.authenticationProvider(daoauth);
```

#### See

- [OWASP Top 10 2021 Category A1](#) - Broken Access Control
- [OWASP Top 10 2017 Category A2](#) - Broken Authentication
- [MITRE, CWE-200](#) - Exposure of Sensitive Information to an Unauthorized Actor

Available In:

**sonarcloud**  | **sonarqube** 