




 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... 🔍

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 "@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

"default" clauses should be last

Code Smell

"equals" method parameters should not be marked "@Nonnull"

Code Smell

A conditionally executed single line should be denoted by indentation

Code Smell

Conditionals should start on new lines

Code Smell

Cognitive Complexity of methods should not be too high

Code Smell

Cipher Block Chaining IVs should be unpredictable

Analyze your code

Vulnerability

Critical

cwe owasp

When encrypting data with the Cipher Block Chaining (CBC) mode an Initialization Vector (IV) is used to randomize the encryption, ie under a given key the same plaintext doesn't always produce the same ciphertext. The IV doesn't need to be secret but should be unpredictable to avoid "Chosen-Plaintext Attack".

To generate Initialization Vectors, NIST recommends to use a secure random number generator.

Noncompliant Code Example

```
public class MyCbcClass {

    public String applyCBC(String strKey, String plainText) {
        byte[] bytesIV = "7cVgr5cbdCZVw5WY".getBytes("UTF-8");

        /* KEY + IV setting */
        IvParameterSpec iv = new IvParameterSpec(bytesIV);
        SecretKeySpec skeySpec = new SecretKeySpec(strKey.getBytes(), "AES");

        /* Ciphering */
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv); // Nonc
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return DatatypeConverter.printBase64Binary(bytesIV)
            + ";" + DatatypeConverter.printBase64Binary(encryptedBytes);
    }
}
```

Compliant Solution

```
public class MyCbcClass {

    SecureRandom random = new SecureRandom();





    public String applyCBC(String strKey, String plainText) {
        byte[] bytesIV = new byte[16];
        random.nextBytes(bytesIV);

        /* KEY + IV setting */
        IvParameterSpec iv = new IvParameterSpec(bytesIV);
        SecretKeySpec skeySpec = new SecretKeySpec(strKey.getBytes(), "AES");

        /* Ciphering */
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv); // Compl
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return DatatypeConverter.printBase64Binary(bytesIV)
            + ";" + DatatypeConverter.printBase64Binary(encryptedBytes);
    }
}
```

https://rules.sonarsource.com/java/RSPEC-3329

1/2

Factory method injection should be used in "@Configuration" classes
 Code Smell
"static" base class members should not be accessed via derived types
 Code Smell
Instance methods should not write to "static" fields
 Code Smell
"indexOf" checks should not be for positive numbers
 Code Smell

```
}  
}
```

See

- [OWASP Top 10 2021 Category A2](#) - Cryptographic Failures
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [MITRE, CWE-329](#) - CWE-329: Not Using an Unpredictable IV with CBC Mode
- [MITRE, CWE-330](#) - Use of Insufficiently Random Values
- [NIST, SP-800-38A](#) - Recommendation for Block Cipher Modes of Operation
- Derived from FindSecBugs rule [STATIC_IV](#)

Available In:



© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
[Privacy Policy](#)