




 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... 🔍


Optional value should only be accessed after calling isPresent()  
 Bug


Overrides should match their parent class methods in synchronization  
 Bug


Value-based classes should not be used for locking  
 Bug


Expressions used in "assert" should not produce side effects  
 Bug


"volatile" variables should not be used with compound operators  
 Bug


"getClass" should not be used for synchronization  
 Bug

Min and max used in combination should not always return the same value  
 Bug

Assignment of lazy-initialized members should be the last step with double-checked locking  
 Bug

"String" calls should not go beyond their bounds  
 Bug

Raw byte values should not be used in bitwise operations in combination with shifts  
 Bug

Getters and setters should be synchronized in pairs  
 Bug

### Using pseudorandom number generators (PRNGs) is security-sensitive

Security Hotspot

Critical

cwe owasp

Using pseudorandom number generators (PRNGs) is security-sensitive. For example, it has led in the past to the following vulnerabilities:

- [CVE-2013-6386](#)
- [CVE-2006-3419](#)
- [CVE-2008-4102](#)

When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.

As the `java.util.Random` class relies on a pseudorandom number generator, this class and relating `java.lang.Math.random()` method should not be used for security-critical applications or for protecting sensitive data. In such context, the `java.security.SecureRandom` class which relies on a cryptographically strong random number generator (RNG) should be used in place.

#### Ask Yourself Whether

- the code using the generated value requires it to be unpredictable. It is the case for all encryption mechanisms or when a secret value, such as a password, is hashed.
- the function you use generates a value which can be predicted (pseudo-random).
- the generated value is used multiple times.
- an attacker can access the generated value.

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

#### Recommended Secure Coding Practices

- Use a cryptographically strong random number generator (RNG) like `java.security.SecureRandom` in place of this PRNG.
- Use the generated random values only once.
- You should not expose the generated random value. If you have to store it, make sure that the database or file is secure.

#### Sensitive Code Example





```
Random random = new Random(); // Sensitive use of Random
byte bytes[] = new byte[20];
random.nextBytes(bytes); // Check if bytes is used for hashi
```

#### Compliant Solution

```
SecureRandom random = new SecureRandom(); // Compliant for s
byte bytes[] = new byte[20];
random.nextBytes(bytes);
```

https://rules.sonarsource.com/java/RSPEC-2245

1/2

|   |
|---|
| <b>Non-thread-safe fields should not be static</b><br> Bug   |
| <b>"null" should not be used with "Optional"</b><br> Bug     |
| <b>Unary prefix operators should not be repeated</b><br> Bug |
| <b>"=+" should not be used instead of "+="</b><br> Bug       |

See

- [OWASP Top 10 2021 Category A2](#) - Cryptographic Failures
- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [Mobile AppSec Verification Standard](#) - Cryptography Requirements
- [OWASP Mobile Top 10 2016 Category M5](#) - Insufficient Cryptography
- [MITRE, CWE-338](#) - Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
- [MITRE, CWE-330](#) - Use of Insufficiently Random Values
- [MITRE, CWE-326](#) - Inadequate Encryption Strength
- [MITRE, CWE-1241](#) - Use of Predictable Algorithm in Random Number Generator
- [CERT, MSC02-J](#) - Generate strong random numbers
- [CERT, MSC30-C](#) - Do not use the rand() function for generating pseudorandom numbers
- [CERT, MSC50-CPP](#) - Do not use std::rand() for generating pseudorandom numbers
- Derived from FindSecBugs rule [Predictable Pseudo Random Number Generator](#)

Available In:

