MSC02-J. Generate strong random numbers

Pseudorandom number generators (PRNGs) use deterministic mathematical algorithms to produce a sequence of numbers with good statistical properties. However, the sequences of numbers produced fail to achieve true randomness. PRNGs usually start with an arithmetic seed value. The algorithm uses this seed to generate an output value and a new seed, which is used to generate the next value, and so on.

The Java API provides a PRNG, the `java.util.Random` class. This PRNG is portable and repeatable. Consequently, two instances of the `java.util.Random` class that are created using the same seed will generate identical sequences of numbers in all Java implementations. Seed values are often reused on application initialization or after every system reboot. In other cases, the seed is derived from the current time obtained from the system clock. An attacker can learn the value of the seed by performing some reconnaissance on the vulnerable target and can then build a lookup table for estimating future seed values.

Consequently, the `java.util.Random` class must not be used either for security-critical applications or for protecting sensitive data. Use a more secure random number generator, such as the `java.security.SecureRandom` class.

Noncompliant Code Example

This noncompliant code example uses the insecure `java.util.Random` class. This class produces an identical sequence of numbers for each given seed value; consequently, the sequence of numbers is predictable.

```
import java.util.Random;
// ...

Random number = new Random(123L);
//...
for (int i = 0; i < 20; i++) {
  // Generate another random integer in the range [0, 20]
  int n = number.nextInt(21);
  System.out.println(n);
}
```

Compliant Solution

This compliant solution uses the `java.security.SecureRandom` class to produce high-quality random numbers:

```
import java.security.SecureRandom;
import java.security.NoSuchAlgorithmException;
// ...

public static void main (String args[]) {
  SecureRandom number = new SecureRandom();
  // Generate 20 integers 0..20
  for (int i = 0; i < 20; i++) {
    System.out.println(number.nextInt(21));
  }
}
```

Compliant Solution (Java 8)

This compliant solution uses the `SecureRandom.getInstanceStrong()` method, introduced in Java 8, to use a strong RNG algorithm, if one is available.

```
import java.security.SecureRandom;
import java.security.NoSuchAlgorithmException;
// ...

public static void main (String args[]) {
   try {
     SecureRandom number = SecureRandom.getInstanceStrong();
     // Generate 20 integers 0..20
     for (int i = 0; i < 20; i++) {
       System.out.println(number.nextInt(21));
     }
   } catch (NoSuchAlgorithmException nsae) {
     // Forward to handler
   }
}
```

Exceptions

MSC02-J-EX0: Using the default constructor for `java.util.Random` applies a seed value that is "very likely to be distinct from any other invocation of this constructor" [API 2014] and may improve security marginally. As a result, it may be used only for noncritical applications operating on nonsensitive data. Java's default seed uses the system's time in milliseconds. When used, explicit documentation of this exception is required.

```
import java.util.Random;
// ...

Random number = new Random(); // Used only for demo purposes
int n;
//...
for (int i = 0; i < 20; i++) {
  // Reseed generator
  number = new Random();
  // Generate another random integer in the range [0, 20]
  n = number.nextInt(21);
  System.out.println(n);
}
```

For noncritical cases, such as adding some randomness to a game or unit testing, the use of class `Random` is acceptable. However, it is worth reiterating that the resulting low-entropy random numbers are insufficiently random to be used for more security-critical applications, such as cryptography.

MSC02-J-EX1: Predictable sequences of pseudorandom numbers are required in some cases, such as when running regression tests of program behavior. Use of the insecure `java.util.Random` class is permitted in such cases. However, security-related applications may invoke this exception *only* for testing purposes; this exception may not be applied in a production context.

Risk Assessment

Predictable random number sequences can weaken the security of critical applications such as cryptography.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
MSC02-J	High	Probable	Medium	**P12**	**L1**

Automated Detection

Tool	Version	Checker	Description
Coverity	7.5	**RISKY_CRYPTO**	Implemented
Parasoft Jtest	10.3	**SECURITY.WSC.SRD**	Implemented
SonarQube	6.7	**S2245**	

Related Vulnerabilities

CVE-2006-6969 describes a vulnerability that enables attackers to guess session identifiers, bypass authentication requirements, and conduct cross-site request forgery attacks.

Related Guidelines

SEI CERT C Coding Standard	MSC30-C. Do not use the rand() function for generating pseudorandom numbers
SEI CERT C++ Coding Standard	MSC50-CPP. Do not use std::rand() for generating pseudorandom numbers
MITRE CWE	CWE-327, Use of a Broken or Risky Cryptographic Algorithm CWE-330, Use of Insufficiently Random Values CWE-332, Insufficient Entropy in PRNG CWE-336, Same Seed in PRNG CWE-337, Predictable Seed in PRNG

Bibliography

[API 2014]	Class `Random` Class `SecureRandom`
[FindBugs 2008]	BC. Random objects created and used only once
[Monsch 2006]