FB.RV\_01\_TO\_INT

Случайное значение от 0 до 1 приводится к целочисленному значению 0. Вероятно, возникнет желание умножить случайное значение на что-то еще, прежде чем приводить его к целому числу, или использовать метод Random.nextInt(n).

Таблица №1 – Основные характеристики детектора FB.RV\_01\_TO\_INT

|  |  |
| --- | --- |
| Категория критичности | Minor |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Numerical Error |
| CWE | CWE-440 |
| CVE | --- |
| Возможные последствия | Неправильная генерация чисел |
| Комментарий (опционально) | Частая ошибка при работе с Random. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Потенциально небезопасное | Корректная конструкция |
| public static void incorrectExample() {  int badRandom = (int)Math.random(); // FB.RV\_01\_TO\_INT  System.out.println("Некорректный случайный номер: " + badRandom); // Всегда 0  } | public static void correctExamples() {  // Способ 1: Math.random с масштабированием  int goodRandom1 = (int)(Math.random() \* 100); // 0-99  System.out.println("Правильный случай 1: " + goodRandom1);    // Способ 2: java.util.Random  int goodRandom2 = new Random().nextInt(100); // 0-99  System.out.println("Правильный случай 2: " + goodRandom2);    // Способ 3: ThreadLocalRandom (Java 7+)  int goodRandom3 = ThreadLocalRandom.current().nextInt(100); // 0-99  System.out.println("Правильный случай 3: " + goodRandom3);    // Способ 4: SecureRandom  int goodRandom4 = new SecureRandom().nextInt(100); // 0-99  System.out.println("Правильный случай 4: " + goodRandom4);  } |

Методы устранения уязвимости при срабатывании детектора

1. Правильное масштабирование;
2. Использование Random.nextInt();
3. ThreadLocalRandom для многопоточности (Java 7+);
4. SecureRandom для криптографии.