FB.RV\_REM\_OF\_HASHCODE

Этот код вычисляет хэш-код, а затем вычисляет остаток этого значения по модулю другого значения. Поскольку hashCode может быть отрицательным, результат операции остатка также может быть отрицательным.

Если необходимо гарантировать, что результат вычислений неотрицателен, может потребоваться изменение кода. Если есть уверенность, что делитель представляет собой степень 2, вместо этого можно использовать побитовый оператор и (т. е. вместо использования x.hashCode()%n использовать x.hashCode()&(n-1)). Это, вероятно, быстрее, чем вычисление остатка.

Если же уверенности нет, что делитель представляет собой степень 2, стоит взять абсолютное значение результата операции с остатком (т. е. использовать Math.abs(x.hashCode()%n)).

Таблица №1 – Основные характеристики детектора FB.RV\_REM\_OF\_HASHCODE

|  |  |
| --- | --- |
| Категория критичности | Minor |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Code Quality Issue |
| CWE | CWE440 |
| CVE | --- |
| Возможные последствия | * Неравномерное распределение значений * Потенциальные коллизии хэшей * Снижение эффективности хэш-структур |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Потенциально небезопасное | Корректная конструкция |
| public static void incorrectTest() {  Object obj = new Object();  // Некорректно: взятие остатка от хэш-кода  int bucket = obj.hashCode() % 100;  System.out.println("Incorrect bucket: " + bucket);  } | public static void correctTest() {  Object obj = new Object();  // Корректно: использование Math.floorMod для равномерного распределения  int bucket = Math.floorMod(obj.hashCode(), 100);  System.out.println("Correct bucket: " + bucket);    // Альтернативный корректный вариант  int bucket2 = (obj.hashCode() & 0x7FFFFFFF) % 100;  System.out.println("Alternative correct bucket: " + bucket2);  } |