FB.RCN\_REDUNDANT\_NULLCHECK\_OF\_NONNULL\_VALUE

Детектор определяет ошибку, в ходе которой метод содержит избыточную проверку известного ненулевого значения на соответствие постоянному нулю.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | CWE476 |
| CVE | — |
| Возможные последствия | Избыточная проверка на null может привести к снижению производительности и усложнению кода. |
| Комментарий | Следует удалить ненужную проверку или убедиться, что переменная действительно не может принимать null, прежде чем убирать проверку. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest(String text) {  if (text == null) {  System.out.println("text равен null");  return;  }  // text гарантированно не null  System.out.println(text.length());  if (text != null) { // Избыточная проверка  System.out.println("Текст: " + text);  }  } | public static void correctTest(String text) {  if (text == null) {  System.out.println("text равен null");  return;  }  System.out.println("Текст: " + text);  } |

Методы устранения уязвимости при срабатывании детектора:

1. **Проанализируйте поток выполнения программы.** Определите, в какой точке кода переменная точно не может быть null.
2. **Удалите все последующие проверки на null для этой переменной, если они находятся в области, где переменная гарантированно ненулевая.**
3. **Убедитесь, что нет путей выполнения, при которых переменная могла бы стать null между моментом, когда она была признана ненулевой, и избыточной проверкой.**
4. Если избыточная проверка кажется необходимой, пересмотрите логику присваивания и предыдущих проверок на null. Возможно, существует ошибка в предыдущей логике,