FB.NP\_NULL\_PARAM\_DEREF\_NONVIRTUAL

Возможно, нулевое значение передается ненулевому параметру метода. Либо параметр помечен как параметр, который всегда должен быть ненулевым, либо анализ показал, что он всегда будет разыменован.

Таблица №1 – Основные характеристики детектора FB.NP\_NULL\_PARAM\_DEREF\_NONVIRTUAL

|  |  |
| --- | --- |
| Категория критичности | Minor |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Null pointer dereference |
| CWE | CWE-476 |
| CVE | --- |
| Возможные последствия | Возможное возникновение NullPointerException |
| Комментарий (опционально) | Особенно критично для библиотечного кода |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Потенциально небезопасное | Корректная конструкция |
| public class NullParamDerefNonVirtual {  // final метод  public final void finalMethod(String input) {  System.out.println(input.trim()); // FB.NP\_NULL\_PARAM\_DEREF\_NONVIRTUAL  }    // static метод  public static void staticMethod(String input) {  System.out.println(input.length()); // FB.NP\_NULL\_PARAM\_DEREF\_NONVIRTUAL  }  } | public class NullParamDerefNonVirtual {  // Защищенный final метод  public final void safeFinalMethod(String input) {  Objects.requireNonNull(input, "Input cannot be null");  System.out.println(input.trim());  }    // Защищенный static метод  public static void safeStaticMethod(String input) {  if (input != null) {  System.out.println(input.length());  }  }  } |

Методы устранения уязвимости при срабатывании детектора

1. Для final методов:

java

Copy

public final void method(final String param) {

Objects.requireNonNull(param, "Param must not be null");

// основная логика

}

1. Для static методов:

java

Copy

public static Optional<String> staticMethod(String param) {

return Optional.ofNullable(param).map(String::trim);

}

1. Для private методов:

java

Copy

private String process(String input) {

assert input != null : "Input parameter";

return input.toLowerCase();

}