FB.TQ\_COMPARING\_VALUES\_WITH\_INCOMPATIBLE\_TYPE\_Q

UALIFIERS

Значение, указанное как содержащее аннотацию квалификатора типа, сравнивается со значением, которое никогда не содержит этот квалификатор.

Точнее, значение, аннотированное квалификатором типа, указывающим, when=ALWAYS, сравнивается со значением, где тот же квалификатор типа указывает, when=NEVER.

Таблица №1 – Основные характеристики детектора FB.TQ\_COMPARING\_VALUES\_WITH\_INCOMPATIBLE\_TYPE\_Q

UALIFIERS

|  |  |
| --- | --- |
| Категория критичности | Minor |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Type-unsafe Comparison |
| CWE | --- |
| CVE | --- |
| Возможные последствия | Ложные результаты сравнения из-за неявного приведения типов; потеря точности при сравнении чисел разного размера; ClassCastException в runtime при сравнении несовместимых объектов; логические ошибки в бизнес-логике программы |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Потенциально небезопасное | Корректная конструкция |
| public static void incorrectTest() {  // 1. Сравнение разных примитивных типов  int intValue = 1000;  long longValue = 1000L;  if (intValue == longValue) { // Неявное приведение int к long  System.out.println("1. Values are equal");  }  // 2. Сравнение объектов разных классов  String str = "1000";  Integer num = 1000;  if (str.equals(num)) { // Всегда false  System.out.println("2. Never happens");  }  // 3. Использование == для объектов  String s1 = new String("text");  String s2 = new String("text");  if (s1 == s2) { // Сравнение ссылок, не содержимого  System.out.println("3. Wrong comparison");  }  // 4. Сравнение @Nullable и @Nonnull  @Nullable String nullable = getNullable();  @Nonnull String nonnull = "text";  if (nullable.equals(nonnull)) { // Возможен NPE  System.out.println("4. Unsafe comparison");  }  } | public static void correctTest() {  // 1. Явное приведение типов  int intValue = 1000;  long longValue = 1000L;  if ((long)intValue == longValue) {  System.out.println("1. Correct type conversion");  }  // 2. Сравнение после преобразования  String str = "1000";  Integer num = 1000;  if (str.equals(num.toString())) {  System.out.println("2. Proper comparison");  }  // 3. Использование equals()  String s1 = new String("text");  String s2 = new String("text");  if (s1.equals(s2)) {  System.out.println("3. Correct object comparison");  }  // 4. Безопасное сравнение с null-проверкой  @Nullable String nullable = getNullable();  @Nonnull String nonnull = "text";  if (nonnull.equals(nullable)) {  System.out.println("4. Safe null-aware comparison");  }  } |

Методы устранения уязвимости при срабатывании детектора.

1. Используйте явное приведение типов перед сравнением (Для примитивов);
2. Всегда используйте equals() вместо ==, проверяйте совместимость типов перед сравнением (Для объектов);
3. Сравнивайте @Nonnull объекты первыми, используйте Objects.equals() (Для null-безопасности);
4. Для чисел разного размера: используйте соответствующие методы сравнения.