FB.WA\_AWAIT\_NOT\_IN\_LOOP

Детектор определяет ошибку, в ходе которой метод содержит вызов java.util.concurrent.await() (или его вариантов), который не находится в цикле. Если объект используется для нескольких условий, то условие, которого вызывающий объект намеревался ожидать, может оказаться не тем, которое произошло на самом деле.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Ожидание (await()) вне цикла может привести к пропуску уведомлений, если условие изменится до вызова метода. |
| Комментарий | Рекомендуется использовать await() внутри цикла, проверяя условие, на которое ожидается изменение. Это гарантирует, что поток возобновит выполнение только после наступления нужного состояния, исключая пропуск уведомлений. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest() {  class Test {  private final Lock lock = new ReentrantLock();  private final Condition condition = lock.newCondition();  public void waitForCondition() throws InterruptedException {  lock.lock();  try {  condition.await(); // await() вне цикла  System.out.println("Condition met, proceeding...");  } finally {  lock.unlock();  }  }  }  } | public static void correctTest() {  class Test {  private final Lock lock = new ReentrantLock();  private final Condition condition = lock.newCondition();  private boolean ready = false;  public void waitForCondition() throws InterruptedException {  lock.lock();  try {  while (!ready) { // проверка условия в цикле  condition.await();  }  System.out.println("Condition met, proceeding...");  } finally {  lock.unlock();  }  }  }  } |

Методы устранения уязвимости при срабатывании детектора:

1. Оберните вызов метода await() (или его вариантов, таких как awaitNanos(), awaitUntil()) в цикл while. Цикл должен проверять фактическое условие, наступления которого ожидает поток.
2. Создайте булеву переменную (или другое состояние), представляющее условие, которое должно быть истинным перед продолжением работы. Доступ к этой переменной должен быть синхронизирован с использованием того же объекта блокировки (Lock).
3. Внутри цикла while вызывайте await() только тогда, когда условие не выполнено. Цикл будет продолжаться до тех пор, пока условие не станет истинным.
4. В потоке, который должен сигнализировать об изменении условия, после изменения состояния переменной условия вызовите методы signal() или signalAll() для соответствующего объекта Condition. Эти вызовы также должны находиться внутри блока, удерживающего связанную блокировку (Lock).
5. Используйте signalAll() вместо signal(), если на одном и том же Condition могут ожидать несколько потоков, и вы хотите разбудить их все для повторной проверки условия.
6. Помните о возможности "ложных пробуждений" (spurious wakeups) при использовании await(). Именно поэтому рекомендуется всегда проверять условие в цикле while после выхода из await().
7. Убедитесь, что доступ к переменной условия и вызовы await(), signal(), signalAll() защищены одной и той же блокировкой (Lock).