FB.TLW\_TWO\_LOCK\_WAIT

Детектор определяет ошибку, в ходе которой ожидание на мониторе, пока удерживаются две блокировки, может привести к взаимоблокировке. Выполнение ожидания снимает только блокировку ожидающего объекта, но не какие-либо другие блокировки. Это не обязательно ошибка, но ее стоит внимательно изучить.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Ожидание (wait()) при удержании двух блокировок может привести к взаимоблокировке, поскольку wait() освобождает только одну из них, а другая остается захваченной. Это может заблокировать выполнение других потоков, ожидающих освобождения второй блокировки, что приведет к зависанию программы. |
| Комментарий | Следует внимательно проверить логику блокировок и по возможности освобождать все ненужные блокировки перед вызовом wait(). Также можно рассмотреть альтернативные механизмы синхронизации, такие как ReentrantLock с Condition. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| private static final Object lock1 = new Object();  private static final Object lock2 = new Object();  public static void incorrectMethod() throws InterruptedException {  synchronized (lock1) {  synchronized (lock2) {  System.out.println("Ожидание...");  // освобождаем только lock1, но lock2 остается захваченной  lock1.wait();  }  }  } | private static final Object lock1 = new Object();  private static final Object lock2 = new Object();  public static void correctMethod() throws InterruptedException {  synchronized (lock1) {  System.out.println("Освобождаем блокировки перед ожиданием...");  }  synchronized (lock2) {  // захватываем, но без риска deadlock  synchronized (lock1) {  lock1.wait();  }  }  } |

Методы устранения уязвимости при срабатывании детектора:

1. **Освобождение всех блокировок, кроме той, на которой вызывается wait():** Перед вызовом метода wait() убедитесь, что текущий поток удерживает только ту блокировку, для которой предназначено ожидание. Освободите все другие захваченные блокировки.
2. **Использование ReentrantLock и Condition:** Класс java.util.concurrent.locks.ReentrantLock предоставляет более гибкие возможности по сравнению со встроенными мониторами (используемыми с synchronized и wait()/notify()). С помощью ReentrantLock вы можете создавать несколько Condition объектов, связанных с одной блокировкой. Ожидание на Condition освобождает только связанную с ним блокировку ReentrantLock, а не другие блокировки, которые мог удерживать поток.
3. **Изменение порядка получения блокировок:** Если возможно, структурируйте код таким образом, чтобы потоки всегда получали несколько блокировок в одном и том же порядке. Это классический способ предотвращения взаимоблокировок. Если все потоки сначала пытаются получить lock1, а затем lock2, вероятность взаимоблокировки снижается. Однако это не всегда возможно и не решает проблему ожидания с удержанными блокировками напрямую.
4. **Избегание вложенных синхронизированных блоков, если это приводит к ожиданию на внешней блокировке:** Постарайтесь реорганизовать код, чтобы избежать ситуаций, когда вы вызываете wait() на блокировке, будучи внутри другого синхронизированного блока.