FB.IJU\_ASSERT\_METHOD\_INVOKED\_FROM\_RUN\_METHOD

Ошибка Детектор определяет ошибку, в ходе которой утверждение JUnit выполняется в методе run. Неудачные утверждения JUnit просто приводят к возникновению исключений. Таким образом, если это исключение возникает в потоке, отличном от потока, вызывающего метод тестирования, исключение завершит поток, но не приведет к сбою теста.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Некорректное поведение тестов, ложноположительные или ложноотрицательные результаты тестирования, потенциальное игнорирование ошибки в многопоточной среде. |
| Комментарий | Вызов методов assert вне метода run в тестовых классах JUnit может привести к тому, что проверка не будет корректно обработана, и тесты не выявят ошибки. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| @Test  public void incorrectTest() throws InterruptedException {  Thread thread = new Thread(  () -> {  // Не выполнится  assertEquals(1, 2);  });  thread.start();  thread.join();  } | @Test  public void correctTest() throws InterruptedException {  final int[] result = new int[1];  Thread thread = new Thread(  () -> {  result[0] = 1;  }  );  thread.start();  thread.join();  // В основном потоке выполнится  assertEquals(1, result[0]);  } |

Методы устранения уязвимости при срабатывании детектора:

1. **Переместите вызовы assert в основной тестовый поток:** Вместо выполнения утверждений внутри потока, запущенного методом run(), передайте результаты работы этого потока обратно в основной тестовый поток и выполните утверждения там.
2. **Используйте механизмы синхронизации:** Применяйте средства синхронизации, такие как CountDownLatch, CyclicBarrier, Future или другие, чтобы дождаться завершения работы дочернего потока перед выполнением утверждений в основном потоке.
3. **Используйте потокобезопасные структуры данных:** Для обмена данными между потоками используйте потокобезопасные коллекции (например, из java.util.concurrent) или атомарные переменные (java.util.concurrent.atomic), чтобы безопасно передавать результаты в основной поток для проверки.
4. **Рассмотрите использование JUnit Rules для асинхронного тестирования:** JUnit предоставляет расширения и правила (Rules) для более удобного тестирования асинхронного кода, которые могут автоматически обрабатывать исключения из дочерних потоков и сигнализировать о сбое теста.
5. **Избегайте прямого вызова assert в run():** Перепроектируйте логику тестирования таким образом, чтобы условия, которые необходимо проверить, оценивались в основном потоке на основе результатов работы асинхронных операций.