FB.OBL\_UNSATISFIED\_OBLIGATION\_EXCEPTION\_EDGE

Этот метод может не выполнить очистку (закрытие, удаление) потока, объекта базы данных или другого ресурса, требующего явной операции очистки.

Как правило, если метод открывает поток или другой ресурс, метод должен использовать блок try/finally, чтобы гарантировать очистку потока или ресурса до возврата метода.

Этот шаблон ошибок по существу аналогичен шаблонам ошибок OS\_OPEN\_STREAM и ODR\_OPEN\_DATABASE\_RESOURCE, но основан на другой технике статического анализа.

Таблица №1 – Основные характеристики детектора FB.OBL\_UNSATISFIED\_OBLIGATION\_EXCEPTION\_EDGE

|  |  |
| --- | --- |
| Категория критичности | Minor |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Resource Leak on Exception Path |
| CWE | --- |
| CVE | --- |
| Возможные последствия | Критические ресурсы не освобождаются при возникновении исключений; накопление утечек при частых ошибках в долгоживущих процессах; взаимоблокировки при исчерпании пулов соединений; коррупция данных при незакрытых транзакциях. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Потенциально небезопасное | Корректная конструкция |
| public static void incorrectTest() {  // 1. Исключение до try-блока  FileInputStream fis = new FileInputStream("data.txt"); // Может выбросить FileNotFoundException  BufferedInputStream bis = new BufferedInputStream(fis); // Может выбросить NullPointerException  try { // Ресурсы могут быть не закрыты при исключениях выше  System.out.println(bis.read());  } finally {  bis.close(); // Не закрывает fis при исключении!  }  // 2. CompletableFuture с потерянными ресурсами  ExecutorService executor = Executors.newSingleThreadExecutor();  Future<File> future = executor.submit(() -> {  File tmp = File.createTempFile("pref", ".tmp");  try (FileOutputStream fos = new FileOutputStream(tmp)) {  fos.write(42); // Если исключение здесь - tmp не удаляется  }  return tmp;  });  executor.shutdown();  } | public static void correctTest() {  // 1. Вложенные try-with-resources  try (FileInputStream fis = new FileInputStream("data.txt");  BufferedInputStream bis = new BufferedInputStream(fis)) {  System.out.println(bis.read());  } catch (IOException e) {  e.printStackTrace();  }  // 2. Управление временными файлами через Cleaner  ExecutorService executor = Executors.newSingleThreadExecutor();  Future<File> future = executor.submit(() -> {  File tmp = File.createTempFile("pref", ".tmp");  tmp.deleteOnExit(); // Гарантированное удаление  try (FileOutputStream fos = new FileOutputStream(tmp)) {  fos.write(42);  }  return tmp;  });    // 3. Дополнительная защита для Future  try {  File result = future.get(1, TimeUnit.SECONDS);  // Обработка результата  } catch (Exception e) {  future.cancel(true); // Прерывание задачи  } finally {  executor.shutdown();  }  } |