HANDLE\_LEAK.FRUGAL.EXCEPTION

Обращает внимание на проблемы управления ресурсами, такие как утечки памяти, утечки файловых дескрипторов или другие типы ресурсов, которые не были корректно освобождены после использования. Такие утечки могут привести к неэффективному использованию ресурсов и повысить риск возникновения проблем с производительностью и надежностью программы.

Преффиск FRUGAL.EXCEPTION может указывать на усиленное внимание к обработке исключений с учетом экономии ресурсов. Это может подразумевать эффективное использование исключений для управления ошибками в коде, предотвращение излишних операций или повторной инициализации ресурсов при возникновении исключительных ситуаций.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | Normal |
| Надежность | Unknown |
| Источник | Roslyn;SvEng |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Может привести к накоплению неосвобождённых ресурсов, снижению производительности и нестабильной работе приложения при длительном выполнении или в условиях ограниченных системных ресурсов. |
| Комментарий | Рекомендуется использовать конструкции, обеспечивающие освобождение ресурсов даже при ошибках, а также минимизировать дублирование операций и повторную инициализацию, особенно в блоках обработки исключений. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest() {  FileInputStream fis = null;  try {  fis = new FileInputStream("test.txt");  int data = fis.read(); // Возможное исключение  System.out.println(data);  } catch (IOException e) {  System.err.println("Ошибка при чтении файла: " + e.getMessage());  } finally {  try {  if (fis != null) {  fis.close(); // Это хорошо, но не покрывает все сценарии  }  } catch (IOException closeEx) {  System.err.println("Ошибка при закрытии файла: " + closeEx.getMessage());  }  }  } | public static void correctTest() {  // Более надёжный способ — try-with-resources  try (FileInputStream fis = new FileInputStream("test.txt")) {  int data = fis.read();  System.out.println(data);  } catch (IOException e) {  System.err.println("Ошибка при чтении или закрытии файла: " + e.getMessage());  }  } |

Методы устранения уязвимости при срабатывании детектора:

1. Используйте конструкцию try-with-resources (начиная с Java 7) для автоматического управления ресурсами, реализующими интерфейс AutoCloseable. Это самый надежный способ гарантировать закрытие ресурсов, даже при возникновении исключений, и предотвращает утечки.
2. Если try-with-resources использовать невозможно, поместите код, работающий с ресурсом, в блок try, а освобождение ресурса - в блок finally. Блок finally выполняется всегда, обеспечивая попытку освобождения ресурса независимо от исключений.