FB.RS\_READOBJECT\_SYNC

Детектор определяет ошибку, в ходе которой сериализуемый класс определяет readObject(), который синхронизируется. По определению, объект, созданный путем десериализации, доступен только одному потоку, поэтому нет необходимости синхронизировать readObject(). Если метод readObject() сам по себе приводит к тому, что объект становится видимым для другого потока, это пример очень сомнительного стиля кодирования.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Избыточная синхронизация в методе readObject() может привести к неоправданным затратам ресурсов и снижению производительности без какой-либо пользы для потокобезопасности. |
| Комментарий | Метод readObject() вызывается в контексте одного потока, поэтому синхронизация внутри него не требуется. Если объект после десериализации должен быть безопасным для многопоточного использования, необходимо применять другие механизмы синхронизации при доступе к нему после восстановления. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest() {  class Test implements Serializable {  @Serial  private static final long serialVersionUID = 1L;  @Serial  private synchronized void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException {  // ненужная синхронизация  stream.defaultReadObject();  }  }  } | public static void correctTest() {  class Test implements Serializable {  @Serial  private static final long serialVersionUID = 1L;  @Serial  private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {  ois.defaultReadObject();  }  }  } |

Методы устранения уязвимости при срабатывании детектора:

1. **Удалите ключевое слово synchronized из объявления метода readObject()**. Синхронизация этого метода не требуется для обеспечения корректной десериализации.
2. **Пересмотрите логику метода readObject()**. Если синхронизация была добавлена с целью обеспечения потокобезопасности объекта после десериализации, это неправильное место для этого. Обеспечение потокобезопасности должно осуществляться при доступе к объекту уже после его полного восстановления. Рассмотрите использование других механизмов синхронизации (например, synchronized блоки в методах, java.util.concurrent классы) при работе с десериализованным объектом в многопоточной среде.
3. **Убедитесь, что процесс десериализации не приводит к преждевременному раскрытию объекта для других потоков до того, как его состояние будет полностью восстановлено.** Такой сценарий является очень необычным и может указывать на серьезную проблему в архитектуре.