FB.SE\_PRIVATE\_READ\_RESOLVE\_NOT\_INHERITED

Детектор определяет ошибку, в ходе которой класс определяет частный метод readResolve. Поскольку он является частным, он не будет унаследован подклассами. Это может быть намеренно и нормально, но следует проверить, чтобы убедиться, что это именно то, что задумано.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Если класс реализует Serializable и содержит приватный метод readResolve(), этот метод не будет унаследован подклассами. Это может привести к тому, что десериализация подклассов создаст новые экземпляры вместо замены десериализованного объекта. В результате могут возникнуть ошибки при работе с синглтонами или другими классами, где важна контроль над десериализацией. |
| Комментарий | Если readResolve() действительно нужен только в базовом классе, можно оставить его private. Однако, если требуется, чтобы подклассы использовали этот метод, следует объявить его как protected или package-private (без private-модификатора). |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| // private readResolve() не наследуется  static class ParentIncorrect implements Serializable {  @Serial  private static final long serialVersionUID = 1L;  @Serial  private Object readResolve() throws ObjectStreamException {  return this;  }  }  static class ChildIncorrect extends ParentIncorrect {  @Serial  private static final long serialVersionUID = 1L;  }  public static void incorrectTest() {  try {  ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("incorrect.ser"));  ObjectInputStream ois = new ObjectInputStream(new FileInputStream("incorrect.ser"));  ChildIncorrect child = new ChildIncorrect();  oos.writeObject(child);  oos.close();  Object deserialized = ois.readObject();  ois.close();  System.out.println(deserialized.getClass().getName());  } catch (IOException | ClassNotFoundException e) {  e.printStackTrace();  }  } | // readResolve() объявлен protected  static class ParentCorrect implements Serializable {  @Serial  private static final long serialVersionUID = 1L;  @Serial  protected Object readResolve() throws ObjectStreamException {  return this;  }  }  static class ChildCorrect extends ParentCorrect {  @Serial  private static final long serialVersionUID = 1L;  }  public static void incorrectTest() {  try {  ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("incorrect.ser"));  ObjectInputStream ois = new ObjectInputStream(new FileInputStream("incorrect.ser"));  ChildIncorrect child = new ChildIncorrect();  oos.writeObject(child);  oos.close();  Object deserialized = ois.readObject();  ois.close();  System.out.println(deserialized.getClass().getName());  } catch (IOException | ClassNotFoundException e) {  e.printStackTrace();  }  } |