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();  }  } |

Методы устранения уязвимости при срабатывании детектора:

1. Определите, должна ли логика замены объектов, реализованная в readResolve() базового класса, применяться к его подклассам.
2. Если подклассы также должны использовать эту логику, измените модификатор доступа метода readResolve() в базовом классе с private на protected или оставьте его без явного модификатора доступа (package-private). protected обеспечивает доступ для подклассов, а package-private - для классов в том же пакете. Выбор зависит от области видимости, в которой вы хотите предоставить доступ к этому методу.
3. Если логика readResolve() специфична только для базового класса и не должна использоваться подклассами, оставьте метод private. В этом случае предупреждение детектора указывает на необходимость явного подтверждения такого намеренного поведения. Возможно, стоит добавить комментарий, объясняющий, почему readResolve() является приватным.
4. Проверьте, не реализуют ли подклассы собственную логику замены объектов через метод readResolve(). Если это так, убедитесь, что эта логика корректна и соответствует требованиям подкласса.
5. Протестируйте процесс сериализации и десериализации как базового класса, так и его подклассов, чтобы убедиться, что объекты восстанавливаются в ожидаемом состоянии и логика readResolve() выполняется корректно (или не выполняется, если это было сделано намеренно).