FB.SE\_METHOD\_MUST\_BE\_PRIVATE

Детектор определяет ошибку, в ходе которой класс реализует интерфейс Serializable и определяет метод пользовательской сериализации/десериализации. Но поскольку этот метод не объявлен закрытым, он будет молча игнорироваться API сериализации/десериализации.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Если методы writeObject(ObjectOutputStream) или readObject(ObjectInputStream) не объявлены как private, механизм сериализации Java их проигнорирует. Это приведёт к использованию стандартной сериализации по умолчанию, что может вызвать потерю данных, нарушение инкапсуляции или потенциальные уязвимости безопасности. |
| Комментарий | Следует объявлять методы writeObject и readObject с модификатором доступа private, чтобы гарантировать их корректный вызов при сериализации и десериализации. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest() {  class A implements Serializable {  private static final long serialVersionUID = 1L;  private String field;  void writeObject(ObjectOutputStream out) throws IOException {  out.writeObject(field);  }  void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {  field = (String) in.readObject();  }  }  } | public static void correctTest() {  class A implements Serializable {  private static final long serialVersionUID = 1L;  private String field;  @Serial  private void writeObject(ObjectOutputStream out) throws IOException {  out.writeObject(field);  }  @Serial  private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {  field = (String) in.readObject();  }  }  } |

Методы устранения уязвимости при срабатывании детектора:

1. Измените модификатор доступа методов writeObject(ObjectOutputStream) и readObject(ObjectInputStream) на private. Убедитесь, что сигнатуры методов точно соответствуют требованиям (возвращаемое значение void, принимаемые аргументы ObjectOutputStream и ObjectInputStream соответственно, и выбрасываемые исключения IOException и ClassNotFoundException).
2. Убедитесь, что в вашем классе действительно требуется пользовательская сериализация/десериализация. Если нет особых требований к обработке сериализации, можно полагаться на механизм по умолчанию, и тогда определение writeObject и readObject не нужно.
3. Проверьте логику внутри методов writeObject и readObject на корректность. Убедитесь, что вы сериализуете и десериализуете все необходимые поля и восстанавливаете состояние объекта правильно.
4. Рассмотрите использование аннотации @Serial (начиная с Java 15) для явного указания, что метод предназначен для сериализации. Хотя это не меняет требования к модификатору доступа private, это может улучшить читаемость кода и помочь инструментам анализа.
5. Протестируйте процесс сериализации и десериализации вашего класса, чтобы убедиться, что пользовательские методы вызываются и объект восстанавливается в ожидаемом состоянии.