FB.IA\_AMBIGUOUS\_INVOCATION\_OF\_INHERITED\_OR\_OUTER\_METHOD

Детектор определяет ошибку, в ходе которой внутренний класс вызывает метод, который может быть преобразован либо в унаследованный метод, либо в метод, определенный во внешнем классе. Например, вы вызываете foo(17), который определен как в суперклассе, так и во внешнем методе. Семантика Java разрешает вызов унаследованного метода, но это может быть не то, что вы хотите.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Использование метода, который может быть разрешен как унаследованный или определенный во внешнем классе, может привести к неопределенному или неожиданному поведению, если вызов не является явным. |
| Комментарий | Явно указывайте, какой метод следует вызвать (например, через super или OuterClass.this), чтобы избежать неоднозначности и обеспечить корректность работы программы. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest() {  class Outer {  public void foo(int x) {  System.out.println("Внешний класс, foo: " + x);  }  }  class Inner extends Outer {  public void foo(int x) {  System.out.println("Внутренний класс, foo: " + x);  }  public void callFoo(int x) {  // Неоднозначный вызов (может вызвать метод как из внешнего, так и из внутреннего класса)  foo(x);  }  }  Inner inner = new Inner();  inner.callFoo(17); // вызов может быть неоднозначным  } | public static void correctTest() {  class Outer {  public void foo(int x) {  System.out.println("Внешний класс, foo: " + x);  }  }  class Inner extends Outer {  @Override  public void foo(int x) {  System.out.println("Внутренний класс, foo: " + x);  }  public void callFoo(int x) {  super.foo(x); // Явное указание на вызов метода внешнего класса  }  }  Inner inner = new Inner();  inner.callFoo(17);  } |