FB.LI\_LAZY\_INIT\_UPDATE\_STATIC

Детектор определяет ошибку, в ходе которой метод содержит несинхронизированную ленивую инициализацию статического поля. После установки поля объект, хранящийся в этом месте, дополнительно обновляется или доступен. Настройка поля видна другим потокам, как только она установлена. Если дальнейшие обращения в методе, устанавливающем поле, служат для инициализации объекта, то у вас очень серьезная ошибка многопоточности, если только что-то еще не мешает любому другому потоку получить доступ к хранимому объекту до тех пор, пока он не будет полностью инициализирован.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | CWE543 |
| CVE | — |
| Возможные последствия | Несинхронизированная ленивую инициализация статического поля с последующими обновлениями объекта может привести к очень серьезным ошибкам многопоточности. |
| Комментарий | Для безопасной ленивой инициализации статического поля следует использовать синхронизацию (например, ключевое слово synchronized, volatile или Double-Checked Locking), чтобы гарантировать, что объект будет полностью инициализирован перед его использованием. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest() {  class Example {  // ...  public void initialize() {  // Дополнительная инициализация объекта  System.out.println("Object initialized");  }  }  class Test {  private static Example instance;  public static Example getInstance() {  // Ленивая инициализация  if (instance == null) {  instance = new Example();  // объект дополнительно обновляется после установки  instance.initialize();  }  return instance;  }  }  } | public static void correctTest() {  class Example {  // ...  public void initialize() {  // Дополнительная инициализация объекта  System.out.println("Object initialized");  }  }  class Test {  private static volatile Example instance;  public static Example getInstance() {  // Ленивая инициализация  if (instance == null) {  synchronized (Test.class) {  if (instance == null) {  instance = new Example();  instance.initialize(); // Дополнительная инициализация  }  }  }  return instance;  }  }  } |

Методы устранения уязвимости при срабатывании детектора:

1. **Оберните создание и всю последующую инициализацию объекта в synchronized блок:** Используйте класс (.class) в качестве монитора для синхронизации.
2. **Используйте статический инициализатор:** Если вся необходимая инициализация может быть выполнена непосредственно при создании объекта, инициализируйте статическое поле в статическом инициализаторе.
3. **Рассмотрите Enum Singleton:** Если вам нужен Singleton, enum обеспечивает потокобезопасную инициализацию.
4. **Избегайте разделения создания объекта и его инициализации в несинхронизированном контексте:** Если объект требует нескольких шагов для полной инициализации, убедитесь, что весь этот процесс происходит атомарно с точки зрения других потоков.
5. **Объявите поле volatile (в сочетании с другими механизмами):** Хотя volatile обеспечивает видимость записи, само создание и инициализация объекта остаются неатомарными. В сочетании с двойной проверкой в synchronized блоке это может быть приемлемым решением.