FB.SC\_START\_IN\_CTOR

Детектор определяет ошибку, в ходе которой конструктор запускает поток. Скорее всего, это будет неправильно, если класс когда-либо будет расширен/подклассифицирован, поскольку поток будет запущен до запуска конструктора подкласса.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Запуск потока в конструкторе может привести к тому, что поток начнет выполнение до полной инициализации объекта, особенно если класс имеет подклассы. |
| Комментарий | Рекомендуется отделить логику запуска потока от конструктора, предоставив отдельный метод для явного старта. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest() {  class Task {  private final Thread thread;  public Task() {  thread = new Thread(() -> System.out.println(Thread.currentThread().getName()));  thread.start(); // поток запускается в конструкторе  }  }  new Task(); // запустится до завершения конструктора  } | public static void correctTest() {  class Task {  private final Thread thread;  public Task() {  thread = new Thread(() -> System.out.println(Thread.currentThread().getName()));  }  public void start() {  thread.start(); // запускается явно, после создания объекта  }  }  Task task = new Task();  task.start(); // запустится после завершения конструктора  } |

Методы устранения уязвимости при срабатывании детектора:

1. Переместите код запуска потока из конструктора в отдельный метод. Создайте новый метод (например, start(), init(), begin()) в вашем классе, который будет отвечать за создание и запуск потока.
2. Вызывайте этот новый метод после создания экземпляра вашего класса. Это гарантирует, что объект будет полностью инициализирован (включая конструкторы всех суперклассов и самого класса) до того, как начнется выполнение потока.
3. Убедитесь, что все необходимые ресурсы и состояния объекта инициализированы до вызова метода запуска потока.
4. Рассмотрите возможность использования фабричных методов или других паттернов создания объектов, которые могут управлять процессом инициализации и запуска потоков более контролируемым образом.
5. Если поток зависит от состояния подкласса, убедитесь, что метод запуска потока вызывается только после создания и полной инициализации экземпляра подкласса.