FB.UG\_SYNC\_SET\_UNSYNC\_GET

Детектор определяет ошибку, в ходе которой класс содержит методы get и set с одинаковыми именами, где метод set синхронизируется, а метод get — нет. Это может привести к некорректному поведению во время выполнения, поскольку вызывающие метод get не обязательно увидят согласованное состояние объекта. Метод get следует сделать синхронизированным.

Таблица №1 – Основные характеристики детектора

|  |  |
| --- | --- |
| Категория критичности | MAJOR |
| Надежность | Unknown |
| Источник | SpotBugs |
| Тип ошибки | Ошибка разработчика |
| CWE | — |
| CVE | — |
| Возможные последствия | Несинхронизированный get()-метод в сочетании с синхронизированным set()-методом может привести к некорректному чтению данных из-за отсутствия гарантии видимости изменений между потоками. |
| Комментарий | Чтобы избежать проблем с конкурентным доступом, метод get() также следует сделать синхронизированным либо использовать volatile для переменной. Альтернативно можно применить java.util.concurrent структуры, такие как AtomicReference или ReadWriteLock. |

Таблица №2 – Примеры срабатывания детектора и устранения уязвимости

|  |  |
| --- | --- |
| Некорректное написание | Корректное написание |
| public static void incorrectTest() {  class Test {  private int value;  public synchronized void setValue(int value) {  this.value = value;  }  public int getValue() {  return value;  }  }  Test test = new Test();  test.setValue(1);  System.out.println(test.getValue());  } | public static void correctTest() {  class Test {  // Volatile гарантирует видимость изменений между потоками  private volatile int value;  public void setValue(int value) {  this.value = value;  }  public int getValue() {  return value;  }  }  Test test = new Test();  test.setValue(1);  System.out.println(test.getValue());  } |

Методы устранения уязвимости при срабатывании детектора:

1. **Синхронизация get()-метода:** Сделайте get()-метод синхронизированным, используя ключевое слово synchronized. Это обеспечит, что при чтении значения переменной будет получена самая последняя запись, сделанная любым потоком, удерживающим монитор объекта.
2. **Использование ключевого слова volatile для переменной:** Объявите общую переменную с ключевым словом volatile. Это гарантирует, что запись в переменную одним потоком будет немедленно видна всем остальным потокам. Однако volatile обеспечивает только атомарность для простых операций чтения и записи. Для составных операций (например, инкремент value++) по-прежнему потребуется синхронизация.
3. **Использование классов из пакета java.util.concurrent.atomic:** Для атомарных операций с простыми типами данных (например, int, long, boolean, reference) можно использовать классы, такие как AtomicInteger, AtomicLong, AtomicBoolean, AtomicReference. Эти классы предоставляют атомарные методы get() и set(), а также другие атомарные операции (например, incrementAndGet(), compareAndSet()).
4. **Использование java.util.concurrent.locks.ReadWriteLock:** Если операции чтения происходят гораздо чаще, чем операции записи, можно использовать ReadWriteLock. Она позволяет нескольким потокам одновременно читать данные (получив "читательскую" блокировку), но обеспечивает эксклюзивный доступ для записи (получив "писательскую" блокировку). Это может повысить производительность в сценариях с большим количеством читателей.