Einführung in Software Engineering

WS 13/14, Aufgabe 1.3

Moritz Nöltner

20. Oktober 2013

1 Dynamisch entdeckbare Softwarefehler

1.1 Out of bounds Zugriff

Greift man auf ein Array zu, und überprüft dabei nicht, ob der Index innerhalb der Arraygrenzen liegt, so kann es sein, dass dieser es nicht ist. In diesm Fall kann je nach Architektur entweder ein Segmentierungsfehler ausgelöst werden, der umliegende Speicher korrumpiert werden, oder wie bei Java eine Out-Out-Bounds-Exception ausgelöst werden.

```java
package tests;
import java.util.Scanner;

public class Main {
    public int[] foo;
    public Main(){
        this.foo=new int[10];
    }

    public static void main(String[] args) {
        Main bar=new Main();
        System.out.println("Bitte geben sie an, welcher Eintrag auf 10"
            + "gesetzt werden soll:");
        Scanner sc = new Scanner(System.in);
        int num = sc.nextInt();
        bar.foo[num]=10;    // Fehler, falls num > 9
        for(int i=0; i<10; i++)
        {
            System.out.println(bar.foo[i]);
        }
    }
}
```

1.2 Nullpointerdereferenzierung

Wenn man beispielsweise die get()-Funktion einer Collection verwendet, kann es vorkommen, dass man als Ergebnis ”null”geliefert bekommt. Vergisst man dann, diesen Fall abzufangen, löst man je nach System einen Segmentierungsfehler aus, greift auf fremden Speicher zu, oder löst eine Exception aus.

2 Statisch entdeckbare Softwarefehler

2.1 Zuweisung statt Vergleich

Oftmals kommt es vor, dass man statt einem logischen Vergleich eine Zuweisung vornimmt.
Beispielsweise:

```
1    ...
2    if(foo = bar) { // statt if(foo == bar) {
3        // do something;
4    }
5    ...
```

2.2 Fehlendes volatile

Wird eine Variable in einem Interrupt verändert, und der Zugriff auf diese Variable nicht mit volatile Synchronisiert, so kann es dazu kommen, dass veraltete Daten weiter genutzt werden. Werden diese dann zurückgeschrieben, kann es sein, dass neuere Daten sogar überschrieben werden.

3 Zusammenfassung

Softwarefehler	**Erklärung**	**Statisch/Dynamisch + Begründung**
Toter Code	Code, der nie ausgeführt wird	Statisch, kann Compiler entdecken
Access out of Bounds	Zugriff auf Array ausserhalb der Arraygrenzen	Dynamisch, wenn man den Index durch Benutzereingabe bekommt
Nullpointerdereferenzierung	Zugriff auf Methoden eines nicht initialisierten Objekts	Dynamisch, weil oft erst zur Laufzeit bekannt ist, ob Objekt null ist.
Vergleich statt Zuweisung	Statt “==“ wird nur “=“ verwendet	Statisch, eigentlich immer sollte in if(...) eine logische Operation stehen.
Verwendung von Java	Es wird Java statt einer “richtigen“ Programmiersprache verwendet	Statisch. Ein guter C++-Compiler erkennt das.
Fehlendes “volatile“	Veränderte Daten werden weitergenutzt	Statisch. Man könnte prüfen, ob auf Daten in einer ISR zugegriffen wird.