DD2460 Software Safety and Security

Introductory Lecture

Dilian Gurov

Lecture Outline

1. The team
2. Introduction to the course
3. Course syllabus
4. Course objectives
5. Course organization

1. Team

- First lecturer: Dilian Gurov
  - E-mail: `dilian@csc.kth.se`
  - Phone: 08-790 81 98 (office)
  - Office: E-bng, floor 4, room 4417
- Second lecturer: Gurvan Le Guernic
  - E-mail: `gurvan@kth.se`
- Course assistant: Musard Balliu
  - E-mail: `musard@kth.se`

2. Software Safety and Security

- **Software safety**:
  - will not damage people or other systems
  - will not fail
- **Software security**:
  - protected against malicious attack
  - data integrity: only authorized access
  - data confidentiality: does not leak information

Software Safety

- A software specification error or design flaw can contribute to or cause a system failure or erroneous human decision
- Example areas:
  - embedded devices like a pacemaker or automotive brake control
  - software controlling nuclear power plants or aerospace rockets
- In summary: no illicit or undesired behaviour

Software Security

- Engineering software so that it continues to function correctly under malicious attack
- Typical **vulnarabilities**:
  - memory leaks
  - buffer overflows
- Vulnarabilities give rise to **threats**

Information Security

- Network security:
  - cryptographic protocols, firewalls, intrusion detection
- Secure information flow:
  - confidentiality, integrity
- Access control:
  - delegation, authorization, trust management

7

Language-based Security

- Application-level attacks:
  - Trojan horses, worms, buffer overrun attacks, exploit attacks, covert channels, and malicious code
- Language-based protection mechanisms:
  - static security analysis
  - program transformation
  - stack inspection

8

Formal Analysis

- **Formal methods**:
  collection of formal notations and techniques (i.e. based on discrete mathematics and mathematical logic) for modelling and analysis of program behaviour.
- **Common goal**:
  The design of *correct* systems.

9

Why *Formal* Methods?

- Only formal methods can capture correctness *precisely*. Basis for *tools*.
- But: formal techniques are *expensive*
- Most needed for:
  - safety-critical systems
  - commercially-critical systems (security)
- Most succesful for: ”small” systems
  - embedded systems
  - communication protocols

10

Formal Verification

- Various techniques
- **Ingredients**:
  - Property class
  - Modelling language
  - Property specification language
  - Verification method (decidability, scalability)
  - Tool support (degree of automation)

11

3. Course Syllabus

- We study three fundamental **techniques** for the analysis of programs, with focus on safety and security.
- The techniques are based on **types** and **logics** for programs, and allow to discover certain types of illicit behaviour or deduce the absence of such behaviour.
- We consider three successful **tools** implementing such techniques.

12

Part I. Temporal Logic and Model Checking

Props: Safety of state sequences
Models: Kripke structures, ProMeLa
Specs: Temporal logic formulas (LTL)
Method: Model checking
Tool: SPIN

13

Part II. Hoare Logic and Program Verification

Props: Safety of data manipulation
Models: Source code (Java) (op. sem.)
Specs: Hoare logic assertions (JML)
Method: VCG, Symbolic execution
Tool: VeriFast

14

Part III. Information Flow Analysis

Props: Confidentiality + integrity of data
Models: Source code (Java)
Specs: Security levels
Method: Type checking
Tool: Jif

15

4. Course Objectives

- **Aim**: provide working familiarity with three methods and tools for the analysis of safety and security of software, in theory and in practice.
- **Grading**: to pass the course, a student has to demonstrate the ability to apply the methods discussed in the course; for the highest grades he/she has also to be proficient in the theoretical underpinnings of these methods.

16

Intended Learning Outcomes

After the course, you should be able to:

1. Identify, specify and verify important safety and security properties using suitable automated tools.
2. Explain the underlying techniques and be able to argue for their correctness and limitations.
3. Correctly interpret and evaluate the results of the analysis.

17

5. Course Organization

- **17 classes**: mixed lectures and tutorials
- **3 lab sessions**: for reporting only!
- **3 take-home assignments**

- Course **web page**:
  `www.csc.kth.se/DD2460/sss12/`

18

Course Literature

- Course **book**:
  ”*Logic in Computer Science*”
  by Huth and Ryan (see Kårbokhandeln)
- Additional material: on the web page
  don’t print without need!

19

Auctioning System

- All three lab assignments are based on the same software system that you will develop: an **auctioning system** written in Java
- The three labs will analyse three different aspects of the system or selected components

20

Lab Assignments

- **SPIN lab**:
  safety of synchronization behaviour
- **VeriFast lab**:
  safety of shared data manipulation
- **Jif lab**:
  confidentiality and integrity of private data

21

Lab Reports

- All three lab assignments will be presented at dedicated lab sessions (in lab room Orange) on the basis of a written report
- The labs will be graded F-C based on
  - quality of work
  - quality of report

22

Tools

- Three tools are installed on the Ubuntu machines in the lab rooms (Orange):
  - **SPIN**: `ispin`
  - **VeriFast**: `vfide`
  - **Jif:** `jif`

23

Take-home Assignments

- Three take-home assignements will examine your understanding of the three analysis techniques and the theoretical underpinnings of the tools
- The assignments will be graded F-A

24