Applied Static Analysis

Software Technology Group
Department of Computer Science
Technische Universität Darmstadt
Dr. Michael Eichberg

Summer Term 2019

Lecture

Every week at 11:40am in S101|A03.

- We are going to discuss algorithms (static analyses) that can be used to detect code smells and security vulnerabilities by analyzing the (binary) code of applications.
- The concepts generally apply to a very wide range of languages and programming language paradigms.
- The examples and analyses are primarily concerned with Java (Bytecode).
- The language that we use for implementing analyses is Scala.

The lecture slides can be found at: https://github.com/stg-tud/apsa.

The lecture slides are basically markdown files that are presented using Deckset. The final PDF is rendered using Marked 2 after preprocessing them using the following sed program:

```
/usr/bin/sed -E -e 's/^\[\..+\:.+\]//' -e 's/^\^[[:space:]]//' -e 's/#[[:space:]]\[fit
\]/#/'
```

The program basically removes all non-standard markdown commands which are only used/required by Deckset.
Hence, if you have suggestions for improvement or if you find typos or more significant issues don't hesitate to create issues. Pull requests can easily be created using the Website and are very welcome.

Exam

- We will have a closed-book exam after the semester.
- The exam will be 60 minutes.
- (There will be no bonus.)

Exercise

Every week we will have ~60minutes of lecture and ~30minutes exercises.

Some exercises will just be theory and some will require you to comprehend and write concrete static analyses. The analyses will be developed using the OPAL framework.

The exercises will help you to prepare for the exam.

Planned Content

- Basic terminology (e.g., soundness, precision, context-sensitivity, ...)
- Code representations (e.g., three-address code)
- Parallelization of static analyses
- Call-graph construction
- Inter-procedural data-flow analyses (IFDS, IDE, ...)
- Purity and immutability analysis
- Escape analysis/Points-to analysis
- Code slicing

Prerequisites

- A keen interest in analyzing code.
- Basic knowledge in compiler construction is helpful.
- A very good understanding of object-oriented programming and in-particular of Java.
- Willing to learn and use Scala.