CSC 699–Static Analysis

Spring 2007
W 5:00-6:00

Instructor Information

Name	:	James Walden
E-Mail	:	waldenj@nku.edu
Office	:	ST 350
Phone	:	(859) 572-5571
Web Site	:	http://www.nku.edu/~waldenj1

Office Hours

MW	1:00-2:00
MW	4:00-5:00
Fr	by appt

Summary

Description	:	Static analysis theory and practice. Program representations, type theory, formal semantics, control flow graphs, data flow analysis, taint checking, and model checking.
Prerequisites	:	CSC 585: Theory of Computation

Student Learning Outcomes

By the end of the course, a successful student should be able to

1. Explain how a programming language is automatically translated to an intermediate representation.
2. Explain how formal semantics can be used to verify security properties of programs.
3. Explain how programs can identify problems in code using control flow and data flow analysis.
4. Use a static analysis tool to identify security vulnerabilities in an application.
5. Design and implement static analysis rules to detect new security vulnerabilities.
6. Evaluate the effectiveness of different static analysis techniques.

Students with Disabilities

Students with disabilities who require accomodations (Academic adjustments, auxiliary aids or services) for this course must register with the Disability Services Office. Please contact the Disability Service Office immediately in the University Center, Suite 320 or call 859-572-6373 for more information. Verification of your disability is required in the Disability Services Office for you to receive reasonable academic accomodations. Visit our website at http://www.nku.edu/~disability/.

Academic Dishonesty

The work that you submit in this course is subject to Northern Kentucky University's Student Honor Code (see `http://www.nku.edu/~deanstudents/student_rights/honor_code.htm`.) Issues involving academic dishonesty are taken very seriously by this instructor and are dealt with according to College and Department policy. Academic dishonesty includes but is not limited to:

1. Improper access to evaluation material or records.
2. Submission of material which is not the student's own work.
3. Conduct which interferes with the work or evaluation of other students.

Some specific examples of dishonesty include:

1. Copying from another person, book, magazine, or other electronic or printed media.
2. Obtaining another person's exam answer or answers.
3. Assisting another student in submitting work that is not the student's own.

It is unacceptable to share program code. It is unacceptable to share homework solutions. It is acceptable and often a good idea to talk about program algorithms and homework solution strategies, but it is not acceptable to use the same code or code segments, or to share actual solutions to homework problems. Any act of academic dishonesty will result in a grade of zero (0) for that item for the first occurrence. An automatic F in the course will result for the second offense. This policy holds for homework assignments and programs, as well as for tests. In order to be fair, penalties will be applied to all parties involved regardless of culpability or fault.

Class Structure and Schedule

1. **Parsing**
   Parsing techniques for programming languages, with a focus on syntax-directed translations using compiler generator tools like `lex` and `yacc`. Program representations, including abstract syntax trees, three-address code, and bytecode.
2. **Type Systems**
   Syntax vs semantic analysis. Types as a simple form of static analysis.
3. **Formal Semantics**
   Formal semantics of the `while` language. The differences between operational, denotational, and axiomatic semantics.
4. **Control Flow Analysis**
   Control flow graphs. Intraprocedural vs inter-procedural analysis.
5. **Data Flow Analysis**
   Data abstraction. Data flow equations. Liveness analysis. Constant propagation. Soundness.

6. **Taint Checking**
   Tracking of dangerous input from source to sink, as provided by Perl and Ruby. Use of type qualifiers in staticly typed languages.

7. **Model Checking**
   Verifying that models of software systems satisifies a specification written in a formal language. Experimentation with model checking systems like Alloy.

8. **Static Analysis**
   Comparative analysis of different types of static analysis tools and frameworks. Use of one or more static analysis frameworks to write a custom analysis.