Course Introduction

Week 1

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

COMP6131 Inaugural Offering at UNSW

Welcome to the inaugural offering of COMP6131 at UNSW!

COMP6131 Inaugural Offering at UNSW

Welcome to the inaugural offering of COMP6131 at UNSW!

- Pre-course survey: `https://forms.gle/r1VLFV8pPmdosPFe7`

COMP6131 Inaugural Offering at UNSW

Welcome to the inaugural offering of COMP6131 at UNSW!

- Pre-course survey: `https://forms.gle/r1VLFV8pPmdosPFe7`
- Give me some key words about your understanding of this course.

COMP6131 Inaugural Offering at UNSW

Welcome to the inaugural offering of COMP6131 at UNSW!

- Pre-course survey: `https://forms.gle/r1VLFV8pPmdosPFe7`
- Give me some key words about your understanding of this course.
- Give me some courses that you think are related.

COMP6131 Inaugural Offering at UNSW

Welcome to the inaugural offering of COMP6131 at UNSW!

- Pre-course survey: `https://forms.gle/r1VLFV8pPmdosPFe7`
- Give me some key words about your understanding of this course.
- Give me some courses that you think are related.
  - System and Software Security Assessment (COMP6447)
  - Security Engineering and Cyber Security (COMP6441/COMP6841)
  - Programming Languages and Compilers (COMP3131/COMP9102)
  - Advanced C++ Programming (COMP6771)
  - Algorithmic Verification (COMP3153/COMP9153)

COMP6131 Inaugural Offering at UNSW

Welcome to the inaugural offering of COMP6131 at UNSW!

- Pre-course survey: `https://forms.gle/r1VLFV8pPmdosPFe7`
- Give me some key words about your understanding of this course.
- Give me some courses that you think are related.
  - System and Software Security Assessment (COMP6447)
  - Security Engineering and Cyber Security (COMP6441/COMP6841)
  - Programming Languages and Compilers (COMP3131/COMP9102)
  - Advanced C++ Programming (COMP6771)
  - Algorithmic Verification (COMP3153/COMP9153)

Your active participation in and off-class discussions, as well as your feedback, will be invaluable. Together, we'll create an engaging and enriching learning experience.

Administration

- Course convenor and lecturer: A/Prof. Yulei Sui
- Email: cs6131@cse.unsw.edu.au
- Course webpage: `https://webcms3.cse.unsw.edu.au/COMP6131/24T2`
- Course forums: `https://edstem.org/au/courses/16128/discussion`
- Important messages will be announced on the course homepage, and urgent messages will also be sent to you via email.
- Course admin:
  - Xiao Cheng (xiao.cheng@unsw.edu.au)
- Lab Demonstrator:
  - Kaiqi Liang (kaiqi.liang@unsw.edu.au)
- Course keywords: Source Code Analysis, Security vulnerabilities, Verification, Abstract Interpretation

Lectures and Labs

- Lecture
  - Time: 11:00 - 13:00, Friday
  - Location: Old Main Building G31 (K-K15-G31)
- Lab
  - Time: 13:00 - 15:00, Friday
  - Location: Old Main Building G32 (K-K15-G32)
- Consultation (appointment is preferred)
  - Time: 15:00 - 16:00, Friday
  - Location: 501H, K17

Course Aim

In this course, you will learn to create **automated code analysis and verification tools** using a **modern compiler** and an **open-source** static analysis framework, to perform **code comprehension**, **vulnerability detection** and code **verification** in real-world software systems.

Teaching Strategy and Rationale

This course has three major components:

- **Lectures** (2 hours per week for 9 weeks)
- **Labs** (2 hours per week for 9 weeks)
- **Assignments** (Assignments 1-3)

This is a project-based course and you are expected to produce a tool towards the end of the course and **NO paper examination** is required!

Teaching Strategy and Rationale

This course has three major components:

- **Lectures** (2 hours per week for 9 weeks)
- **Labs** (2 hours per week for 9 weeks)
- **Assignments** (Assignments 1-3)

This is a project-based course and you are expected to produce a tool towards the end of the course and **NO paper examination** is required!

Assessment Type	Name	Percentage %
Lab work	Quiz-1 & Exercise-1	10%
	Quiz-2 & Exercise-3	10%
	Quiz-3 & Exercise-3	10%
Assignment-1	Information flow tracking	20%
Assignment-2	Symbolic execution	25%
Assignment-3	Abstract interpretation	25%

Lectures

Course contents:

- Foundational **theories of static code analysis and verification** aimed at detecting bugs and verifying the absence of bugs.
- Some practical **demonstrations of examples and coding**
- **Problem-solving skills** (algorithms, testing, debugging)
- Lecture slides typically available before each lecture

Lectures

Course contents:

- Foundational **theories of static code analysis and verification** aimed at detecting bugs and verifying the absence of bugs.
- Some practical **demonstrations of examples and coding**
- **Problem-solving skills** (algorithms, testing, debugging)
- Lecture slides typically available before each lecture

Get the most out of COMP3131:

- **Attend the lectures/labs and get involved!** Ask questions in class and on Ed forums (no pasting code solutions allowed).
- **Be open-minded**. While you will use C++ to implement your code checker for analyzing C code in this course. Consider developing a code checker using the learned theories within a modern compiler setting.
- **Research and development mentality**. Keep your curiosity to learn the most recent/advanced source code analysis techniques in this course.

Labs

Labs: Hands-on experience includes **preparatory activities** and building the **skills needed for assignments** through completing

- three set of **quizzes**
- three **coding exercises** (small-scale)

Labs

Labs: Hands-on experience includes **preparatory activities** and building the **skills needed for assignments** through completing

- three set of **quizzes**
- three **coding exercises** (small-scale)

Submission and marking

- Done **individually**
- Submitted a single cpp file by uploading to WebCMS or via `give`.
- Automarked (with manual checks and partial marking) against our internal tests.

Assignments

Three assignments: Each assignment is built on the previous one to develop code-checking tools capable of:

- Assignment-1: tracking tainted information flows
- Assignment-2: performing symbolic execution
- Assignment-3: conducting abstract interpretation

Assignments

Three assignments: Each assignment is built on the previous one to develop code-checking tools capable of:

- Assignment-1: tracking tainted information flows
- Assignment-2: performing symbolic execution
- Assignment-3: conducting abstract interpretation

Best practice for completing an assignment (e.g., Assignment-1)?

- **Correct way**: Lab-Quiz-1 $\rightarrow$ Lab-Exercise-1 $\rightarrow$ Assignment-1
- **Incorrect way**: all other orders

Assumed Knowledge

- Should have
  - Experience in writing, debugging, and testing programs in C (COMP2521, COMP9024).
  - Experience working on a large programming project (assignment).
  - Knowledge of using Git and programming IDEs like VSCode.
  - Willingness to learn and open-mindedness.
- Nice to have
  - Some knowledge of object-oriented programming (you will get more practice in C++ programming in our labs).
  - Some background knowledge of compilers (e.g., LLVM and SVF).
  - Some knowledge of secure coding.
  - Experience at different programming ”levels” (e.g. low-level, high-level).

Learning Outcomes

- Practice **system programming skills** to develop **code analysis and verification techniques** to address code security and reliability problems.
  - **High-quality coding**: Commit to writing high-quality, error-free, and high-performance code, especially within the context of large-scale codebases.
  - **Compiler basics**: Gain insights into compilation, code representation, low-level instructions, code debugging and profiling.
  - **Vulnerability Comprehension**: Understand common vulnerabilities, such as tainted information flow, buffer overflows, and assertion errors.
  - **Open-source static analysis framework**: learn to build practical tools on top of open-source frameworks like SVF.
  - **Formal Verification**: Understand formal methods and techniques for verifying code correctness using mathematical and logical reasoning tools.

Learning Outcomes

- Practice **system programming skills** to develop **code analysis and verification techniques** to address code security and reliability problems.
  - **High-quality coding**: Commit to writing high-quality, error-free, and high-performance code, especially within the context of large-scale codebases.
  - **Compiler basics**: Gain insights into compilation, code representation, low-level instructions, code debugging and profiling.
  - **Vulnerability Comprehension**: Understand common vulnerabilities, such as tainted information flow, buffer overflows, and assertion errors.
  - **Open-source static analysis framework**: learn to build practical tools on top of open-source frameworks like SVF.
  - **Formal Verification**: Understand formal methods and techniques for verifying code correctness using mathematical and logical reasoning tools.
- Career and job roles
  - Software Engineer; Security Analyst/Engineer; Compiler Engineer; Formal Methods Engineer; Software Reliability Engineer; Embedded Systems Developer; Research Scientist (in Academia or Industry);

Course Schedule

Week	Content	Lab & Assignment Start	Due (23:59, Wednesday)
1	**Lecture**: Course Overview and Introduction **Lab**: C++ practices, vulnerability assessment and compiler IR	Quiz-1 + Exercise-1 (10%)	-
2	**Lecture**: Control and Data Flows **Lab**: Code graphs, SVF, constraints solving	Assignment 1 (20%)	-
3	**Lecture**: Pointer Aliasing and Taint Tracking **Lab**: Tainted information flow tracking	-	Quiz-1 + Exercise-1
4	**Lecture**: Code Verification Basis **Lab**: Verification concepts, predicate logic	Quiz-2 + Exercise-2 (10%)	Assignment-1
5	**Lecture**: Automated Theorem Proving **Lab**: Manual assertion-based verification using Z3	Assignment 2 (25%)	-
6	**Flexibility Week**	-	-
7	**Lecture**: Code Verification using Symbolic Execution **Lab**: Automated code assertion verification using Z3	-	Quiz-2 + Exercise-2
8	**Lecture**: Abstract Interpretation Foundations **Lab**: Basic concepts and examples	Quiz-3 + Exercise-3 (10%)	Assignment-2
9	**Lecture**: Code Verification using Abstract Interpretation **Lab**: Manual assertion-based verification using Z3	Assignment 3 (25%)	-
10	**Lecture**: Buffer Overflow Detection using Abstract Interpretation **Lab**: Implementation and testing	-	Quiz-3 + Exercise-3 Assignment-3 (Week 11)

Assessment Guidelines

- **Joint Work Prohibited**: Collaboration on this assignment is not allowed. This is strictly an individual task.
- **Individual Submission**: The work you submit must be entirely your own. Submitting any work, even partially, written by someone else is prohibited.
- **Assignment Examination**: Submissions will be scrutinised both automatically and manually for external authorship.
- **Prohibition on Sharing Work**: Sharing, publishing, or distributing your assignment is not allowed even after the course ends. Do not share your work with anyone other than the COMP6131 teaching staff. Do not publish your lab or assignment code online (e.g., on a public GitHub repository), as they may be used by future students.

Violation of these conditions may result in an academic integrity investigation. For more information, read the UNSW Student Code.

Marking and Plagiarism

- Please refer to specs for each assessment before you start at WebCMS
- No extension is allowed and late submission is strongly discouraged.

Marking and Plagiarism

- Please refer to specs for each assessment before you start at WebCMS
- No extension is allowed and late submission is strongly discouraged.
- The UNSW standard late penalty for assessment is 5% per day for 5 days - this is implemented hourly for this assignment. Your assignment mark will be reduced by 0.2% for each hour (or part thereof) late past the submission deadline.
  - For example, if an assignment worth 60% was submitted half an hour late, it would be awarded 59.8%, whereas if it was submitted past 10 hours late, it would be awarded 57.8%.
- Beware - submissions 5 or more days late will receive zero marks. This again is the UNSW standard assessment policy.
- The marking results will normally be released one week after each submission deadline.

Plagiarism: see course outline for penalties

Course Materials and Resources

No single textbook covers all the course content. Recommended references and an abundance of online materials are available below:

- Static Value-Flow Analysis Framework for Source Code
  - https://github.com/SVF-tools/Teaching-Software-Security-Analysis
  - https://github.com/SVF-tools/SVF
- Compilers: Principles, Techniques, and Tools Hardcover, https://www.amazon.com.au/Compilers-Alfred-V-Aho/dp/0321486811
- LLVM Compiler https://llvm.org/
- Symbolic Execution https://en.wikipedia.org/wiki/Symbolic_execution
- Abstract Interpretation https://en.wikipedia.org/wiki/Abstract_interpretation
- Z3 Theorem Prover
  - https://github.com/Z3Prover/z3
  - https://theory.stanford.edu/~nikolaj/programmingz3.html