

Course Introduction

Week 1

Yulei Sui

School of Computer Science and Engineering
University of New South Wales, Australia

COMP6131 at UNSW

Welcome to COMP6131!

- Pre-course survey: <https://forms.gle/eTLjzg7GR7Y88qy2A>

COMP6131 at UNSW

Welcome to COMP6131!

- Pre-course survey: <https://forms.gle/eTLjzg7GR7Y88qy2A>
- Give me some key words about your understanding of this course.

COMP6131 at UNSW

Welcome to COMP6131!

- Pre-course survey: <https://forms.gle/eTLjzg7GR7Y88qy2A>
- Give me some key words about your understanding of this course.
- Give me some courses that you think are related.

COMP6131 at UNSW

Welcome to COMP6131!

- Pre-course survey: <https://forms.gle/eTLjzg7GR7Y88qy2A>
- 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)
 - Software Testing and Quality Assurance (COMP3142)

COMP6131 at UNSW

Welcome to COMP6131!

- Pre-course survey: <https://forms.gle/eTLjzg7GR7Y88qy2A>
- 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)
 - Software Testing and Quality Assurance (COMP3142)

Your active participation in and off-class discussions, as well as your feedback, will be invaluable. Hope to make your learning experience an enjoyable and rewarding one.

Who this course is for / not for

- This is not a pure tool-driven security course to find vulnerabilities.
- This course is about reasoning, and understanding **why** they exist, **when** they can or cannot occur, and **what assumptions** our analyses rely on.

Who this course is for / not for

- This is not a pure tool-driven security course to find vulnerabilities.
- This course is about reasoning, and understanding **why** they exist, **when** they can or cannot occur, and **what assumptions** our analyses rely on.
- Good fit if you enjoy:
 - **reasoning** about programs, understanding the guarantees and limits
 - **compilers intermediate representation** and program semantics
 - how **static analysis** tools work under the hood
 - thinking about **programs abstractly** (e.g., all possible executions)
 - **develop an open-source tool yourself** to reason and detect vulnerabilities

Who this course is for / not for

- This is not a pure tool-driven security course to find vulnerabilities.
- This course is about reasoning, and understanding **why** they exist, **when** they can or cannot occur, and **what assumptions** our analyses rely on.
- Good fit if you enjoy:
 - **reasoning** about programs, understanding the guarantees and limits
 - **compilers intermediate representation** and program semantics
 - how **static analysis** tools work under the hood
 - thinking about **programs abstractly** (e.g., all possible executions)
 - **develop an open-source tool yourself** to reason and detect vulnerabilities
- Not a good fit if you expect:
 - pentesting / exploit labs
 - fast, surface-level tools without theory
 - it's not focused on quickly getting “yes/no” answers from black-box systems
 - not a course about learning lots of security tools
 - those are valuable skills — just not the focus here.

Administration and Important Course Links

- Course convenor and lecturer: Yulei Sui
- Email: cs6131@cse.unsw.edu.au
- Course webpage: <https://cgi.cse.unsw.edu.au/~cs6131>
- Lab-Exercise/Assignment specifications, and code templates:
<https://github.com/SVF-tools/Software-Security-Analysis>
- Course forums (Discourse), login with your zID (not email) and zPass:
<https://discourse03.cse.unsw.edu.au/26T2/COMP6131>
- Important messages will be announced on the course homepage or via email.
- Course admin: Jiawei Wang (jiawei.wang6@unsw.edu.au)
- Lab demonstrators and tutors
 - Mohamad Barbar (m.barbar@unsw.edu.au)
 - Hanyuan Li (hanyuan.li@unsw.edu.au)
 - Angela He (angela.he@student.unsw.edu.au)
- Course keywords: Static Analysis and Verification, Security Vulnerabilities, Control- and Data-Flows, Symbolic Execution, Abstract Interpretation

Lectures and Labs

- Lecture
 - Time: 14:00 - 16:00, Friday
 - Location: **Electrical Engineering G23 (K-G17-G23)**
- Lab
 - Time: 16:00 - 18:00, Friday
 - Location: **NS Global 127 (Webster 127) (K-G14-127)**
- Helper Session (CSE Help, K17 Ground Floor)
 - Time: 11:00 - 12:00, Tuesday

Lecture first, lab straight after — continuity in one afternoon: learn it, try it, and debug it before it escapes.

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 reasoning**, **vulnerability detection** and code **verification** in real-world software systems.

Teaching Strategy and Rationale

This course has three major components:

- **Lectures** (10 weeks excluding Week 6 for study break)
- **Labs** (10 weeks excluding Week 6 for study break)
- **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 (at least this year)!

Teaching Strategy and Rationale

This course has three major components:

- **Lectures** (10 weeks excluding Week 6 for study break)
- **Labs** (10 weeks excluding Week 6 for study break)
- **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 (at least this year)!

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	20%
Assignment-3	Abstract interpretation	30%

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 are recorded.

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 are recorded.

Get the most out of COMP6131:

- **Attend the lectures/labs and get involved!** Ask questions in class and on course forums (no pasting code solutions allowed).
- **Open-minded to other languages.** While you use C++/Python to implement your code checker for C programs in this course. Consider developing a 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. Labs are typically not recorded and we may try to record some demonstrations.

- **Lab session:** demo, specs and solutions walkthrough, tutor interaction, and hands-on Q&A (bring your own laptop).
- three set of **quizzes** (multiple choice questions)
- three **coding exercises** (small-scale)

Labs

Labs: Hands-on experience includes **preparatory activities** and building the **skills needed for assignments** through completing. Labs are typically not recorded and we may try to record some demonstrations.

- **Lab session:** demo, specs and solutions walkthrough, tutor interaction, and hands-on Q&A (bring your own laptop).
- three set of **quizzes** (multiple choice questions)
- three **coding exercises** (small-scale)

Submission and marking

- Done **individually**
- Submitted a single cpp file in each lab exercise 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: developing 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: developing abstract interpretation

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

- **Correct way:** Lab-Quiz-1 $\leftrightarrow$ Lab-Exercise-1 $\rightarrow$ Assignment-1

Assumed Knowledge

- Should have
 - Experience in writing, debugging, and testing programs in C (COMP2521, COMP9024).
 - Knowledge of using Git and programming IDEs like vim or VSCode.
 - Willingness to learn and open-mindedness.
- Nice to have
 - Some knowledge of object-oriented programming (e.g., C++) and/or Python, which is useful for using the SVF library and completing assignments)
 - 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 Assessment**: Understand common vulnerabilities, such as tainted information flow, buffer overflows, null dereferences, 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 Assessment**: Understand common vulnerabilities, such as tainted information flow, buffer overflows, null dereferences, 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
 - DevSecOps 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: programming practices, graph algorithms, vulnerabilities	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 (20%)	-
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 (30%)	-
10	Lecture: Bug Detection using Abstract Interpretation Lab: Implementation and testing	-	Quiz-3 + Exercise-3 Assignment-3 (Week 11)

Assessment Guidelines

- **Assessment Specs Vary Yearly:** Follow the latest lab/assignment specs and **start only after** they're **announced and released** on WebCMS.
- **Joint Work Prohibited:** Collaboration on this assignment is not allowed. Each quiz, exercise, and assignment is submitted and marked individually.
- **Individual Submission:** The work you submit must be entirely your own. Submitting any work, even partially, written by someone else is prohibited.
- **Assessment Marking:** Submissions will be examined 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).**

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 - For 5 or more days late submissions, will receive zero marks. This again is the UNSW/CSE assessment policy. Any new submissions (including ELS/special cases) after the feedback by tutors in labs are not allowed.
- The marking results will normally be released around one week after each submission deadline.

Plagiarism: see [course outline for penalties](#)

CSE Student Reps and Concerns

You are not alone!

What to do in the event of a problem or concern with the course

1

In the first instance, please try to resolve the issue with the **immediate party** - which in most cases will be your **tutor**

2

If unresolved, please escalate to the **course admins and lecturer**

3

If unresolved, please escalate to the CSE Student Representatives at **stureps@cse.unsw.edu.au** (or anonymously through our website)

4

If unresolved, please escalate to the CSE Grievance Officers at **grievance-officer@cse.unsw.edu.au**

5

If unresolved, please escalate to **UNSW Complaints** via the UNSW website

Brought to you by the CSE Student Representatives

Find us at <https://cgi.cse.unsw.edu.au/~stureps/>

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>