Proposal-82594 Software Security Analysis and Verification

MARK AS COMPLETE

Proposal Creation PROPOSED

Proposal Rationale

Proposal Overview

New / Revise	New
Proposal Summary and Rationale *	Software systems, which are deeply rooted in a wide range of industries and businesses, are so pervasive that we are often unaware of their presence until software bugs occur. A single bug can cause critical software failures, resulting in huge social and economic impacts. Despite the increasing attention and efforts in improving software reliability and security, modern complex systems (e.g. containing millions of lines of code) are still plagued with bugs. This subject aims to develop automated software analysis and verification techniques based on an open-source framework to understand and discover common yet important software vulnerabilities in system software. Through this subject, students will be given the opportunity to apply and practice their system programming skills and software development experience. Particularly, the students will design and develop automated code analysis tools to understand, discover and detect programming errors to improve software quality. Students are assessed on the basis of their technical capabilities, understanding of software analysis and verification via open-source software development. The course proposal has been prepared over the past two months and discussed with DHoS John Shepherd. The rationale behind this proposed course is twofold: (1) an increasing number of CSE students specializing in software engineering require a course that caters to the demands of secure software engineering and programming languages. This elective course aims to fulfil those needs. (2) The proposed course has also been discussed in CSE's Future Cybersecurity Working Group to address the requirements of the planned cybersecurity program, particularly in the area of secure coding. The proposed course is planned to be included in the Cybersecurity Engineering Masters program (MIT), which may start in 2024 as endorsed by the faculty.

Proposal Contacts

Author	Yulei Sui

Proposal Sponsor	Name	Role
	John Shepherd	Senior Lecturer
Collaborators	Name	Role
	Gianna Foong	Education Quality Officer

Consultation

Consultations	Who	Relationship
	Salil Kanhere	Internal to UNSW
	Paul Hunter	Internal to UNSW

Third Party Arrangement

Third Party Delivery Partners/Organisations	

COMPXXXX Software Security Analysis and Verification : Overview

MARK AS COMPLETE

2024.01 PROPOSED

Related Items Report

Course Information

Overview

Course Code *	COMPXXXX
Course Name *	Software Security Analysis and Verification
Course Name - SiMs *	Code Analysis and Verification
Owning Faculty *	Faculty of Engineering
Owning Academic Unit *	School of Computer Science and Engineering
Collaborating Academic Unit	
Administrative Campus *	Sydney
Units of Credit *	6
Grading Basis *	Standard UNSW grades
Academic Calendar Type *	3+
Career *	Postgraduate

Academic Details

Course Description for Handbook *	This course is designed to provide a systematic exploration of automated source code analysis and verification techniques, with the aim of gaining hands-on experience in implementing code analysis tools to identify

	common yet important software vulnerabilities in software systems. By taking this course, students can put static analysis theories and advanced techniques into practice. They will be able to create static code analysis tools (e.g., written in C++) based on modern compilers and popular open-source frameworks to scan, comprehend and detect programming mistakes and vulnerabilities with the purpose of enhancing code quality and security.
Field of Education (Broad) *	020000 Information Technology
Field of Education (Narrow) *	020100 Computer Science
Field of Education (Detailed) *	020103 Programming
Level	
Teaching Strategies and Rationale	The lectures in this course provide a theoretical foundation for static code analysis and verification techniques, which aim to detect and defend against software vulnerabilities that can result in unexpected behaviors or exploitation by attackers. The lectures also explain how to apply these techniques to create practical tools for detecting and reporting common software vulnerabilities and proving the absence of bugs in system code (such as code written in C/C++). The tutorials build upon the lectures to solidify students' conceptual understanding of software vulnerabilities, static analysis algorithms, and associated tools, including modern compilers, low-level intermediate representation of code, and constraint-solving techniques. The tutorials also provide practical knowledge relevant to the week's lab exercises. The laboratory exercises give students a concrete understanding of compiler-based source code analysis and verification. They develop automatic auditing tools, identify vulnerabilities, and develop security analysis and testing methods to improve code quality and protect systems against cyber-attacks, which are the core conceptual outcomes of this course. Programming assignments expand upon the skills and knowledge developed in laboratory exercises by tackling substantial practical problems. Each assignment builds upon the previous one to finally create a tool that is ready to be used for source code scanning and vulnerability detection.
Course Aims	The primary goal of this course is to familiarize students with a variety of source code analysis techniques and algorithms, ranging from fundamental to state-of-the-art. Students will be encouraged to develop their own tools based on an open-source framework that uses a compiler, and they will learn how these techniques and tools can be applied to detect real-world code vulnerabilities. Upon completing the course, students will have a comprehensive understanding of the history of source code analysis, the current techniques used, and the future challenges that must be addressed in this field.
Delivery Attributes	

Course Properties

Course Type	Award course
Course Attributes	
Repeat for Credit	No

Delivery

Delivery Variations	

Learning Outcomes

HELP

Learning Outcomes

Code	Description
CLO1	Explain source code vulnerabilities in system software and motivations for static analysis and verification

Code	CLO1
Description *	Explain source code vulnerabilities in system software and motivations for static analysis and verification
Number *	1

Code	Description
CLO2	Understand basic compiler intermediate representation and its importance for precise software vulnerability detection

Code	CLO2
Description *	Understand basic compiler intermediate representation and its importance for precise software vulnerability detection
Number *	2

Code	Description
CLO3	Implement static code analysis techniques including control- and data-flow analysis

Code	CLO3
Description *	Implement static code analysis techniques including control- and data-flow analysis
Number *	3

Code	Description
CLO4	Develop static code verification techniques including constraint solving and assertion-based verification using automated theorem provers.

Code	CLO4
Description *	Develop static code verification techniques including constraint solving and assertion-based verification using automated theorem provers.

Number *	4

CLO5	Design and implement well-considered, high performance, static analysis algorithm to solve problems

Code	CLO5
Description *	Design and implement well-considered, high performance, static analysis algorithm to solve problems
Number *	5

CLO6	Create effective and minimum unit tests and documentation to validate the correctness of the code analysis tools

Code	CLO6
Description *	Create effective and minimum unit tests and documentation to validate the correctness of the code analysis tools
Number *	6

Course Mapping

CLO Mapping

Mapping ☐ Show versions

No mapping data exists

Assessments

Assessments

Assessment Type	Assessment Name	Weighting (%)
Lab work	Weekly coding exercises and quizzes	30

Assessment Type *	Lab work
Assessment Name *	Weekly coding exercises and quizzes
Weighting (%) *	30
Group or Individual *	Individual
Assessment Overview *	During weekly labs, students will be given programming exercises and quizzes to complete as part of their assessment to enhance in-class learning and get prepared for code assignments. These tasks will cover topics such as understanding compiler intermediate representation, graph representation of code, implementing basic graph traversal algorithms, and using Z3 theorem prover for assertion verification. These tasks will provide students with the opportunity to revisit and solidify individual concepts from the lectures. The small-scale coding tasks will be marked based on the

	correctness of the student's implementation, and in-person feedback will be provided to help students improve their coding skills, and understanding of the key knowledge in lectures.
Mapping to Learning Outcomes	CLO1, CLO2, CLO3, CLO4, CLO5, CLO6
Applies to All Delivery Variations?	Yes
Assessment Number *	1
Assessment Mapping	> Mapping

Assignment	Control-flow reachability analysis	20

Assessment Type *	Assignment
Assessment Name *	Control-flow reachability analysis
Weighting (%) *	20
Group or Individual *	Individual
Assessment Overview *	The objective of this assignment is to create a control-flow reachability analysis using a code graph representation, with the purpose of automatically analyzing a program's properties (such as control-flows) and its execution states. The implementation of this assignment is a component of a course-generated automated source code analysis tool. The assignment requires students to enhance their coding and debugging skills by utilizing online slides and resources. These resources will also provide opportunities to learn how to write efficient and high-quality code checkers. As part of practicing the fundamental programming elements, the student is required to submit their implementation, which should compile correctly and produce the desired outputs. This assignment's implementation enhances the student's comprehension of source code analysis and code representation, concurrently serving as preparation for the development of a software analysis tool in future assessments.
Mapping to Learning Outcomes	CLO2, CLO3
Applies to All Delivery Variations?	Yes
Assessment Number *	2

Assessment Mapping	Mapping

Assignment	Data dependence and taint tracking	25

Assessment Type *	Assignment
Assessment Name *	Data dependence and taint tracking
Weighting (%) *	25
Group or Individual *	Individual
Assessment Overview *	The objective of this assignment is to create a data-dependence analysis and a tainted information flow tracker that integrates both the control-flow reachability analysis (in Assignment 1) and data-dependence information. This enables the tracking of tainted input and the identification of tainted information flows on the code graph. The implementation of this assignment is a component of a course-generated automated source code analysis tool. In this assignment, students are expected to grasp the concepts of data dependence and pointer analysis and apply them practically to construct a tainted information tracker. Alongside practicing fundamental programming elements, students are required to submit their implementations, which should compile without errors and generate the intended outputs.
Mapping to Learning Outcomes	CLO2, CLO4
Applies to All Delivery Variations?	Yes
Assessment Number *	3
Assessment Mapping	Mapping

Assignment	Assertion-based code verification	25

Assessment Type *	Assignment
Assessment Name *	Assertion-based code verification
Weighting (%) *	25
Group or Individual *	Individual
Assessment Overview *	The objective of this assignment is to utilize static symbolic execution to develop assertion-based verification,

		which automates the verification of code properties through the use of assertions. The assertion verification builds upon the weekly lab exercises as well as the control- and data-flow analysis implemented in Assignments 2 and 3, incorporating the control-flow reachability and data dependence information. In this assignment, students are tasked with practicing the utilization of the satisfiability modulo theories solver and applying it to automated assertion verification. By submitting accurate implementations that yield the desired outputs, students will successfully complete the final component of the automated source code analyzer for the course. This component enables the analysis and verification of real-world C programs.
	Mapping to Learning Outcomes	CLO4, CLO5, CLO6
	Applies to All Delivery Variations?	Yes
	Assessment Number *	4
	Assessment Mapping	Mapping
Assessment Total Percentage	COMPXXXX	100
Requirements to pass this course		

Enrolment Requirements and Relationships

Enrolment Requirements	Type		Description	Career
	Enrolment Requirements		COMP6771	Postgraduate
	Type *	Enrolment Requirements		
	Description	COMP6771		
	Career *	Postgraduate		
Additional Constraints on Enrolment Requirements or Relationships				
Course Relationships				

Third Party Arrangements

Third Party	

Resourcing	
Revenue Split	
Other Information for Handbook	
Key Search Terms	
Links to Course Outline	
Academic Admin Only	
Student System ID	
Course Active	Yes
Publication Flag	Yes
Requisites	

Course content:

Designed based on my past open courses in code security analysis and verification.
https://github.com/SVF-tools/Teaching-Software-Analysis/wiki
https://github.com/SVF-tools/Teaching-Software-Verification/wiki

Analysis and comprehension of code vulnerabilities*: Identifying memory safety errors, buffer overflows, memory leaks, use-after-frees, null dereferences, and information leakage.*

LLVM Compiler and Intermediate Representation*: Low-level representation of code, compiler instructions and the translation of high-level code to low-level instructions.*

SVF Static Analysis Framework*: Graphs representation of code, encompassing control-flow graphs, call graphs, data-flow graphs, and constraint programming.*

Control-Flow analysis*: Context-insensitive and context-sensitive reachability analysis on top of interprocedural control-flow graphs.*

Data-Flow Analysis*: This analysis focuses on tracking information flow and performing taint analysis to detect information leakage, considering sensitive sources and sinks within the control-flow and data-flow graphs.*

Code verification using Z3 Theorem prover*: Abstracting code into first-order logic expressions and validating the satisfiability of constraints using the Z3 Theorem prover.*

Symbolic execution and assertion-based checking*: Conducting abstract execution on the control-flow graph to compute and maintain program states. Verify code properties such as assertions for each program execution path.*

Relationship with other courses (Most of the following courses are undergraduate courses):

COMP3153/9153 Algorithmic Verification *emphasis on the theoretical aspects of verification, specifically pertaining to logics, automata, verification games, and model checking. It does not focus extensively on code security or tool implementation. My proposed course primarily focuses on the practical application side, utilizing the automated Z3 Theorem Prover.*

COMP2111: System Modelling & Design *is a theoretical course that encompasses propositional, predicate, and Hoare logics, proof obligations, and proofs, without delving into code implementation.*

COMP6447 System and Software Security Assessment *concentrates on industry practices and security assessment, covering topics such as cyber-attacks, exploitation, and defense. However, it does not cover code analysis and verification.*

COMP4161 Advanced Topics in Software Verification *is closely linked to specification-driven verification and proof, with a particular focus on interactive theorem provers like Isabelle. This course does not emphasis on static analysis or compiler-based code checkers that aim to detect vulnerabilities in real-world programs.*