CS 6301-002: Language-based Security

Course Information

Title: CS/SE 6301-002: Language-based Security
Course Registration Number: 82035/82036
Times: MW 1:00–2:15
Location: ECSN 2.120
Instructor: Dr. Kevin Hamlen (hamlen AT utdallas)
Instructor's Office Hours: MW 2:15–3:15, ECSS 3.704

Course Summary

This course will introduce and survey the field of Language-based Software Security, in which techniques from compilers and programming language theory are leveraged to address issues in computer security. Topics include:

1. Certifying Compilers
2. In-lined Reference Monitors
3. Software Fault Isolation
4. Address Space Randomization
5. Formal Methods
6. Web Scripting Security
7. Information Flow Control

The aim of the course is to allow each student to develop a solid understanding of at least one of these topics, along with a more general familiarity with the range of research in the field. In-course discussion will highlight opportunities for cutting-edge research in each area. If you do research involving software security, this course will provide you with an array of powerful tools for addressing software security issues. If you do research involving programming languages or compilers, this course will show you how to take techniques that you already know and apply them to a new and important problem domain. If your career involves management or development of high-assurance software systems, this course will provide a comparative analysis of traditional versus language-based techniques.

The course is open to Ph.D. students and Masters students. Interested undergraduates should see the instructor for permission to take the course.

Suggested (non-mandatory) prerequisite: CS 6371 Advanced Programming Languages (or taken concurrently)

Grading

Homework (30%): For the first 10 weeks of the course, students will complete a series of programming exercises assigned through eLearning. Background material helpful for completing the exercises can be found in the online textbook Software Foundations.

Quizzes (30%): Most classes will begin with a short quiz testing the students comprension of an assigned reading for the day. Questions will typically be multiple choice or short answer. The easier questions will be designed to test whether the student has read the material, and the harder ones will test deeper understanding of more subtle points.

Class Participation (10%): Students are expected to come to class having read the assigned paper(s), and prepared with questions, critiques, and discussion topics. Regular attendance and class participation will count 10% towards their grades in the course.

Project (30%): Students will work individually or in a small team for the last 6 weeks of the course to complete a small project using the Coq theorem-prover. A typical project will involve implementing and formally verifying a standard algorithm of the student's choosing. Students will present their projects in class, with the presentation counting toward their project grade.

Texts

In our study of the Coq theorem proving system, we will be using the following online textbook:

- Benjamin C. Pierce, Chris Casinghino, Michael Greenberg, Vilhelm Sjöberg, and Brent Yorgey. Software Foundations, Version 4.0. University of Pennsylvania, May 2016.

For those who wish to explore Coq in greater depth (e.g., for developing their projects), the following book by the Coq developers is highly recommended:

- Yves Bertot and Pierre Castéran. Interactive Theorem Proving and Program Development. Springer-Verlag, 2004.

Additionally, the following two texts available through the UTD library may be useful for general background on type theory and computer security, respectively:

- Benjamin C. Pierce, ed., Advanced Topics in Types and Programming Languages. MIT Press, Cambridge, MA 2005. (available online from UTD computers)

- Matt Bishop. Computer Security: Art and Science. Addison-Wesley, 2003. (available online from UTD computers)

Tentative Course Schedule

Date	Topic	Assigned Reading(s)	Coq Exercises
Program-Proof Co-development			
Lecture 1: Mon 8/22	**Introduction to Formal Methods and Secure Software Development**		
Lecture 2: Wed 8/24	**Higher-order Types**	Software Foundations: "Basics" chapter, up to and including the first two exercises (nandb, andb3).	
Lecture 3: Mon 8/29	**Machine-verified Proofs**	F. Williams. Investigating SANS/CWE Top 25 Programming Errors. Tech. Rep. ICTN 6870, E. Carolina University, 2009.	
Lecture 4: Wed 8/31	**Proof Tactics**	J. Walden, J. Stuckman, and R. Scandariato. Predicting Vulnerable Components: Software Metrics vs Text Mining. In *Proc. 25th Int. Sym. Software Reliability Engineering (ISSRE)*, pp. 23–33, November 2014.	Assignment 1 due 8/31 (Coq Basics)
No Class: Mon 9/5	**No Class:** Labor Day		
Lecture 5: Wed 9/7	**Logical Operators**	P. Hudak. Conception, Evolution, and Application of Functional Programming Languages. ACM Computing	Assignment 2 due 9/7 (Induction)

		Surveys, 21(3):359–411, May 1989. • **Required sections:** Abstract, Introduction, 2.1, 2.3, and 2.4	
Lecture 6: Mon 9/ 12	**Constructivistic Logic** Veridrone Source Repository on GitHub	G. Malecha, D. Ricketts, M.M. Alvarez, and S. Lerner. Towards Foundational Verification of Cyber-physical Systems, Invited Paper. In *Proc. Science of Security for Cyber-Physical Systems Workshop (SoSCyPS)*, April 2016.	Assignment 3 due 9/14 (Lists)
Lecture 7: Wed 9/ 14	**Non-termination, Soundness, and Contradiction**	no assigned reading	
Lecture 8: Mon 9/ 19	**Coq Wrap-up** FCF Source Repository on GitHub	A. Petcher and G. Morrisett. The Foundational Cryptography Framework. In *Proc. Int. Conf. Principles of Security and Trust (POST)*, 2015.	Assignment 4 due 9/21 (Polymorphism)
Software Security Fundamentals			
Lecture 9: Wed 9/ 21	**The Science of Security**	J. Bau and J. C. Mitchell. Security Modeling and Analysis. IEEE Security & Privacy 9(3):18–25, 2011.	
Lecture 10: Mon 9/ 26	**Formally Verified Compilation**	X. Leroy. Formal Verification of a Realistic Compiler. Communications of the ACM, 52(7):107–115, 2009.	Assignment 5 due 9/28 (Tactics & Logic)

Lecture 11: Wed 9/ 28	**Machine Code Validation**	• D. Brumley, I. Jager, T. Avgerinos, and E.J. Schwartz. BAP: A Binary Analysis Platform. In *Proc. Int. Conf. Computer Aided Verification (CAV)*, 2011. • A Formal Specification for BIL: BIL Instruction Language, October 2015.	
Lecture 12: Mon 10/ 3	**Software Model-checking**	M. Müller-Olm, D. Schmidt, and B. Steffen. Model-Checking: A Tutorial Introduction. In *Proc. 6th Int. Sym. Static Analysis (SAS)*, pp. 330–354, September 1999. • **Required sections:** 1–4.2 and 5 • I will not quiz you on the parts about fixed point theory in Section 3.3, but do read the paragraph on Computational Tree Logic.	
Code-reuse Attacks and Defenses			
Lecture 13: Wed 10/ 5	**Return-oriented Programming**	E.J. Schwartz, T. Avgerinos, and D. Brumley. Q: Exploit Hardening Made Easy. In *Proc. 20th USENIX Security Symposium*, 2011.	Assignment 6 due 10/10 (Inductive Propositions)
Lecture 14:	**Artificial Diversity**	H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N.	

Mon 10/ 10		Modadugu, and D. Boneh. On the Effectiveness of Address-Space Randomization. In *Proc. ACM Conf. Computer and Communications Security (CCS)*, pp. 298–307, 2004.	
Lecture 15: Wed 10/ 12	**Control-flow Integrity**	M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti. Control-Flow Integrity: Principles, Implementations, and Applications. In *Proc. ACM Conf. Computer and Communications Security*, pp. 340–353, 2005.	
Lecture 16: Mon 10/ 17	**Binary Stirring**	R. Wartell, V. Mohan, K.W. Hamlen, and Z. Lin. Binary Stirring: Self-randomizing Instruction Addresses of Legacy x86 Binary Code. In *Proc. ACM Conf. Computer and Communications Security (CCS)*, 2012.	
Lecture 17: Wed 10/ 19	**Binary Software Security Retrofitting**	R. Wartell, V. Mohan, K.W. Hamlen, and Z. Lin. Securing Untrusted Code via Compiler-Agnostic Binary Rewriting. In *Proc. 28th Annual Computer Security Applications Conf. (ACSAC)*, pp. 299–308, December 2012.	
Lecture 18: Mon 10/ 24	**TBA**	G. Morrisett, G. Tan, J. Tassarotti, J.-B. Tristan, and E. Gan. RockSalt: Better, Faster, Stronger SFI for the x86. In *Proc. 33rd ACM SIGPLAN Conf. Programming Languages Design and Implementation*	Assignment 7 due 10/26 (Case Study)

		(PLDI), pp. 395–404, June 2012.	
Lecture 19: Wed 10/ 26	**Project Group Meetings**	no assigned reading	
In-lined Reference Monitors			
Lecture 20: Mon 10/ 31	**Theory of IRMs**	F.B. Schneider. Enforceable Security Policies. ACM Transactions on Information and System Security, 3(1):30–50, 2000.	Project
Lecture 21: Wed 11/ 2	**Aspect-Oriented Programming and IRMs**	M. Jones and K.W. Hamlen. Disambiguating Aspect-oriented Security Policies. In *Proc. 9th Int. Conf. Aspect-Oriented Software Development (AOSD)*, pp. 193–204, March 2010.	
Lecture 22: Mon 11/ 7	**Model-checking IRMs**	K.W. Hamlen, M.M. Jones, and M. Sridhar. Aspect-oriented Runtime Monitor Certification. In *Proc. 18th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 126–140, March–April 2012.	
Lecture 23: Wed 11/ 9	**Language-based Web Scripting Security**	P.H. Phung, M. Monshizadeh, M. Sridhar, K.W. Hamlen, and V.N. Venkatakrishnan. Between Worlds: Securing Mixed JavaScript/ActionScript Multi-party Web Content. IEEE Transactions on Dependable and Secure Computing (TDSC),	

		12(4):443–457, July–August 2015.	
Information Flow Controls			
Lecture 24: Mon 11/ 14	**Intro to Information Flow**	A. Sabelfeld and A.C. Myers. Language-Based Information-Flow Security. IEEE J. Selected Areas in Communications, 21(1):5–19, 2003.	
Lecture 25: Wed 11/ 16	**Type-based Information Flow Controls**	A.C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *Proc. 26th ACM Sym. Principles of Programming Languages (POPL)*, pp. 228–241, 1999.	
No Class: Mon 11/ 21	**No Class:** Fall Break		
No Class: Wed 11/ 23	**No Class:** Fall Break		
Software Cyber Deception			
Lecture 26: Mon 11/ 28	**Honeypots and Honey-patching**	F. Araujo, K.W. Hamlen, S. Biedermann, and S. Katzenbeisser. From Patches to Honey-Patches: Lightweight Attacker Misdirection, Deception, and Disinformation. In *Proc. ACM Computer and Communications Security (CCS)*, 2014.	
Lecture 27:	**Dynamic Secret Redaction**	F. Araujo and K.W. Hamlen. Compiler-instrumented, Dynamic Secret-Redaction	

Wed 11/ 30		of Legacy Processes for Attacker Deception. In *Proc. USENIX Security Symposium,* 2015.	
Lecture 28: Mon 12/ 5	**TBA**		
Lecture 29: Wed 12/ 7	**Project Presentations**		