CS 6301-002: Language-based Security

Dr. Kevin Hamlen
Fall 2017

Prerequisites: none(?)

In order to accommodate a certain celestial event, class will start at 1:15 today.
Outline

• Course logistics
  – course objectives
  – homework grading, etc.
  – about me
• What is “Language-based Security”?  
• Tentative course schedule (list of topics)
• Demo: Program-proof co-development
Course Information

• Course webpage:
  – http://www.utdallas.edu/~hamlen/cs6301fa17.html
  – google “hamlen”, click top link, scroll to bottom

• Instructor:
  – Dr. Kevin Hamlen
  – ECSS 3.704
  – Office hours: After class (MW 2:15-3:15)
Course Objectives

• Cutting-edge research
  – Read historic and modern research articles
  – Learn about modern efforts toward a science of computer security
  – Learn basics of programming language theory, functional programming, automated theorem-proving, etc.
  – Get your hands dirty: Implement and formally verify something

• Warning: This is a research-level class!
  – Many problems/questions are open-ended. We will be exploring the known issues together.
  – Not only is the software extremely beta, the whole concept behind the software is extremely beta!
Grading

• Homework (30%)
  – programming exercises – learn to program in Coq
  – **first one ("Basics") due next Wednesday 8/30**
  – see online schedule for the other six due dates
  – Recommendation: Complete them far in advance! Then you’ll be done!
  – If you have trouble, do some exercises in the online text (Pierce et al.)

• Quizzes (30%)
  – start of each class (~15 min.)
  – covers assigned reading for the day
  – first one next Monday

• Class participation (10%)
  – discuss article, ask questions

• Projects (30%)
  – implement and/or verify a language-based security defense
  – project proposals due around mid-semester (tentatively 11/1)
  – implement during last 6 weeks of course

• No exams
Quizzes

• Approximately 8 questions each
  – multiple-choice / short answer
• Difficulty level
  – multiple-choice != obvious-choice
  – main concepts (e.g., “What is this paper (really) about?”)
  – feasibility critique: main limitations, pros/cons
  – a few harder in-depth questions to test whether you caught subtle core details
• Warning: These articles are HARD to understand!
  – contain many tiny technical details
  – I don’t test on minutiae. Don’t memorize everything.
  – But hard questions might focus on a seemingly minor item that turns out to be very significant.
Comprehending Papers

• Ability to read and digest research articles (at a reasonable pace) is a learned and very valuable skill.
  – articles are extremely dense!
  – most assume background knowledge that you lack
  – I expect you to look up terms you don’t understand on your own initiative.
  – I don’t expect you to understand everything, even after doing your best to look things up.

• After reading, be sure you can answer the following:
  – What’s the MAIN discovery?
  – Why is this better/worse than alternatives?
  – What are the system’s weaknesses? How can I break it?
  – Do you understand the main definitions / notations?
About Me

• originally from the northeastern US (Buffalo, NY)

• Undergrad
  – Carnegie Mellon (computer science and math)
  – Senior thesis: Proof-Carrying Code

• Masters (’02) & Ph.D. (’06)
  – Cornell (computer science)
  – Dissertation: certifying in-lined reference monitors

• Industry experience
  – Microsoft Research (Redmond & Cambridge)
  – language-based security for .NET and F#

• Personal
  – married, 4-year-old son, 2-year-old twin boys
  – Christian (Protestant)
What is LBS?

• Leveraging theory of programming language design and compiler construction to enforce software security

• Two domains of research:
  – new languages/tools for creating secure software from scratch
  – securing legacy code (e.g., written in C)

• Three stages of enforcement
  – static (find & fix vulnerabilities before runtime)
  – dynamic (detect and block attacks at runtime)
  – audit (recover and assign blame after an attack)
Grand Challenge: Secure Program Development

• How can you ever be sure that your software is 100% correct and immune to attack?

• Scenario: You are hired to write the control software for a nuclear reactor.
  – it must NEVER fail (millions of lives at stake)
  – it must cope with adversarial conditions (prime target)
  – it must be efficient (too slow = meltdown)

• Traditional approaches
  – test a lot (“It didn’t crash today…”)
  – write a proof (consisting of about 10K pages of math)
    • How do we know there isn’t a bug in the proof??
Grand Challenge: Securing Legacy Code

• Scenario: NSA wants secure software on their office workstations.
  – need web browsers, document readers, etc.
  – need internet connectivity
  – stores and/or reads top secret documents
  – not feasible to rebuild the entire universe of software from the ground up
  – software is proprietary (and usually closed-source)

• How to stop secrets from leaking?
Grand Challenge: A Science of Security

• Can we develop a science of security like we have for math or physics?
  – Are there iron-clad “proofs” of security?
  – What does it even mean for a system to be “secure”?
  – Are there metrics for security? Can we determine that one system is “more secure” than other? Can we prove that it’s “80% secure”?
  – Are there some security policies that are provably unenforceable? Can we prove that certain enforcement mechanisms can enforce certain classes of policies and not others?
Tentative List of Topics

• First 4 weeks:
  – Developing machine-verified software with Coq
  – basis for homework and some projects
• Next 2 weeks: LBS foundations
• After that, move into cutting-edge research:
  – Software Model-checking
  – Software Fault Isolation
  – Code-injection and code-reuse attacks & defenses
  – Artificial Software Diversity and Obfuscation
  – Cyber offense (google “hamlen Frankenstein”)
  – Information flow controls (confidentiality enforcement)
  – Web scripting security
  – In-lined Reference Monitoring
  – Attacker Deception (google “hamlen heartbleed”)
Coq: Programming with Proofs

• Coq
  – stands for “Calculus of Constructions” (the underlying type theory of the system)
    • named after mathematician Thierry Coquand
  – developed by INRIA, France over last decade
  – most powerful secure software development system to date (in my opinion)
• Specification language based on ML/OCaml
  – all loops are recursive (no while/for loops)
  – immutable variables (variables are assign-once!)
  – first-class functions
  – parametrically polymorphic
  – higher-order, dependent type system (!)
• Demo
Homework

• Download and install Coq
  – see links to Coq page from course web page
  – use version 8.6

• Read for next time:
  – “Preface” of the Software Foundations online text (see course web page).
  – Read the “Basics” chapter up to first exercise
  – Solve first two exercises (nandb, andb3)