CS260 – Advanced Systems Security

Introduction

March 31, 2025

About Me

- *Trent Jaeger* (PhD, University of Michigan)
- Professor since 2005, CSE -- after 9 years at IBM Research
- Research: Operating System Security
- Example Systems
  - L4 Microkernel – Minimal, high-performance OS
  - Linux – Open source, UNIX variant
  - Xen hypervisor – Open source, virtual machine platform
  - Cloud systems – OpenStack – Open source, IaaS cloud platform
- Office: WCH 442; Hours: W 10-11 and by appt
- Email: trentj@ucr.edu

This course….

- Is a systems course that teaches principles for building a secure system and techniques for implementing those principles
  - Caveat: We are still trying to figure out the latter
  - Topics: What makes a system secure (principles); Example implementations of such principles (at OS, VMM, application, etc.); Challenges in building secure systems; Tools to assist in implementations; Recent research in secure systems design

Background

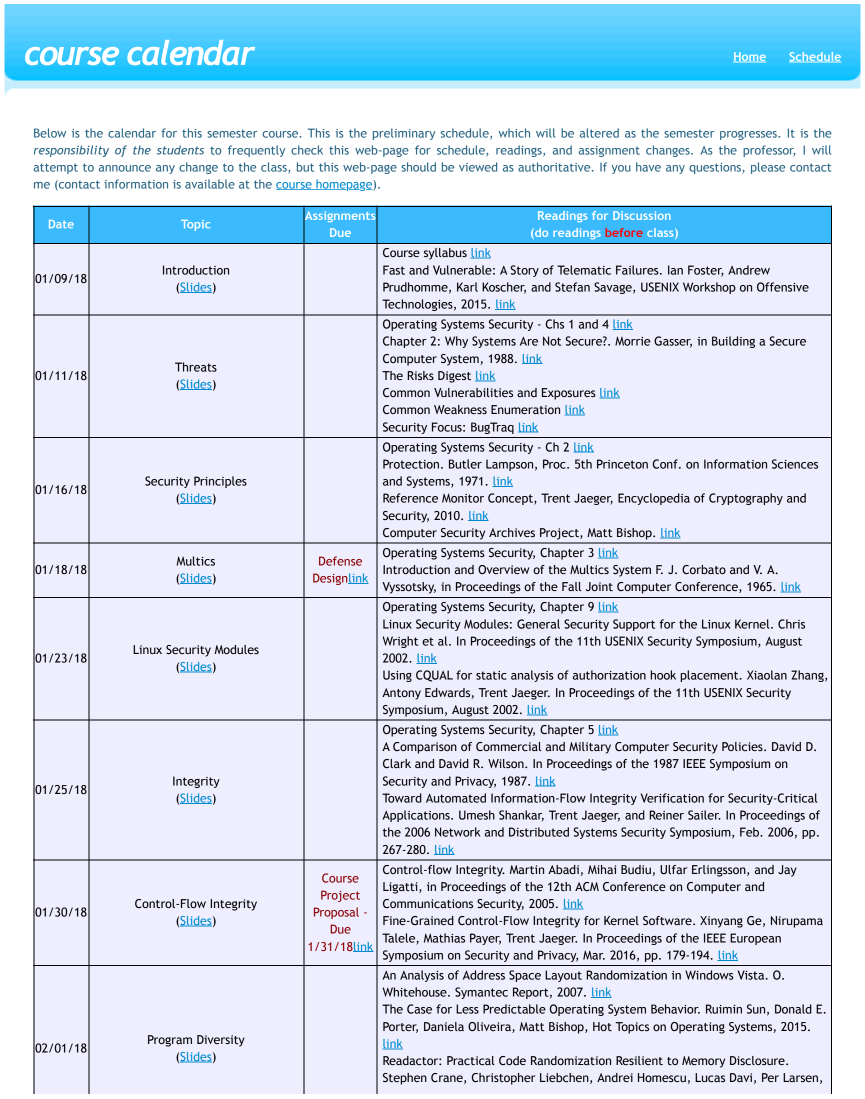
- Required:
  - CS 202 (Adv OS) or CS 255 (Adv Security)
- Expected:
  - Assume solid OS and software background
- Additional:
  - Willingness to read
    - We are going to read a lot of systems security papers
  - Willingness to program
    - We are going to have a course project

Course Materials

- Course Website
  - http://www.cs.ucr.edu/~trentj/cs260-s25/
  - Course assignments, slides, etc. will be placed on the linked schedule site
    - Check back often -- papers/assignments may change
- Course Textbook
  - My book: *Operating Systems Security*
    - *Available for free from Springer– Google “Operating Systems Security, Trent Jaeger”*
      - *https://link.springer.com/book/10.1007/978-3-031-02333-0*
  - Augmented with research papers and CyBoK reports

Course Calendar

- The course calendar has all the details
- Links to online papers for readings
- Links to slides
- Please check the calendar frequently
  - It's the real-time state of the course



course calendar

Home Schedule

Below is the calendar for this semester course. This is the preliminary schedule, which will be altered as the semester progresses. It is the *responsibility of the students* to frequently check this web-page for schedule, readings, and assignment changes. As the professor, I will attempt to announce any change to the class, but this web-page should be viewed as authoritative. If you have any questions, please contact me (contact information is available at the course homepage).

Date	Topic	Assignments Due	Readings for Discussion (do readings before class)
01/09/18	Introduction (Slides)		Course syllabus link Fast and Vulnerable: A Story of Telematic Failures. Ian Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage, USENIX Workshop on Offensive Technologies, 2015. link
01/11/18	Threats (Slides)		Operating Systems Security - Chs 1 and 4 link Chapter 2: Why Systems Are Not Secure?. Morrie Gasser, in Building a Secure Computer System, 1988. link The Risks Digest link Common Vulnerabilities and Exposures link Common Weakness Enumeration link Security Focus: BugTraq link
01/16/18	Security Principles (Slides)		Operating Systems Security - Ch 2 link Protection. Butler Lampson, Proc. 5th Princeton Conf. on Information Sciences and Systems, 1971. link Reference Monitor Concept, Trent Jaeger, Encyclopedia of Cryptography and Security, 2010. link Computer Security Archives Project, Matt Bishop. link
01/18/18	Multics (Slides)	Defense Designlink	Operating Systems Security, Chapter 3 link Introduction and Overview of the Multics System F. J. Corbato and V. A. Vyssotsky, in Proceedings of the Fall Joint Computer Conference, 1965. link
01/23/18	Linux Security Modules (Slides)		Operating Systems Security, Chapter 9 link Linux Security Modules: General Security Support for the Linux Kernel. Chris Wright et al. In Proceedings of the 11th USENIX Security Symposium, August 2002. link Using CQUAL for static analysis of authorization hook placement. Xiaolan Zhang, Antony Edwards, Trent Jaeger. In Proceedings of the 11th USENIX Security Symposium, August 2002. link
01/25/18	Integrity (Slides)		Operating Systems Security, Chapter 5 link A Comparison of Commercial and Military Computer Security Policies. David D. Clark and David R. Wilson. In Proceedings of the 1987 IEEE Symposium on Security and Privacy, 1987. link Toward Automated Information-Flow Integrity Verification for Security-Critical Applications. Umesh Shankar, Trent Jaeger, and Reiner Sailer. In Proceedings of the 2006 Network and Distributed Systems Security Symposium, Feb. 2006, pp. 267-280. link
01/30/18	Control-Flow Integrity (Slides)	Course Project Proposal - Due 1/31/18link	Control-flow Integrity. Martin Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti, in Proceedings of the 12th ACM Conference on Computer and Communications Security, 2005. link Fine-Grained Control-Flow Integrity for Kernel Software. Xinyang Ge, Nirupama Talele, Mathias Payer, Trent Jaeger. In Proceedings of the IEEE European Symposium on Security and Privacy, Mar. 2016, pp. 179-194. link
02/01/18	Program Diversity (Slides)		An Analysis of Address Space Layout Randomization in Windows Vista. O. Whitehouse. Symantec Report, 2007. link The Case for Less Predictable Operating System Behavior. Ruimin Sun, Donald E. Porter, Daniela Oliveira, Matt Bishop, Hot Topics on Operating Systems, 2015. link Readactor: Practical Code Randomization Resilient to Memory Disclosure. Stephen Crane, Christopher Liebchen, Andrei Homescu, Lucas Davi, Per Larsen,

Course Communications

- Canvas
  - I would you send me emails directly
    - trentj@ucr.edu
  - Please use “CS260” in the subject
- I will create a slack account for the course
  - Seems to be used sparingly
  - Send me an email if I do not respond within a day

Project Options

- Research Project Areas
- Software security
  - E.g., Memory safety defenses for C/C++ and Rust
- Linux security
  - eBPF enforcement and sandboxing extensions
- Android security
- File systems security
- Security kernel (seL4) use

Grading

- Exam (35%)
  - One exam
    - In class - week 8
- Course Project (40%)
  - Research on an operating systems security topic
  - Project presentation
- Others (25%)
  - Readings and reviews
  - Participation
  - Homework(s)
  - Announced, in-class quizzes (maybe)

Academic Integrity

- See UCR’s Policy on Academic Integrity
  - https://conduct.ucr.edu/policies/academic-integrity-policies-and-procedures

Ethics Statement

- This course considers topics involving personal and public privacy and security. As part of this investigation, we will cover technologies whose abuse may infringe on the rights of others. As an instructor, I rely on the ethical use of these technologies. Unethical use may include circumvention of existing security or privacy measurements for any purpose, or the dissemination, promotion, or exploitation of vulnerabilities of these services. Exceptions to these guidelines may occur in the process of reporting vulnerabilities through public and authoritative channels. Any activity outside the letter or spirit of these guidelines will be reported to the proper authorities and may result in dismissal from the class.
- When in doubt, please contact the instructor for advice. **Do not** undertake any action which could be perceived as technology misuse anywhere and/or under any circumstances unless you have received explicit permission from Professor Jaeger.

Road Map

- Introduction
  - 1. Threats 2. Why do we need OS security?
- Vulnerabilities
  - 1. Memory safety   2. Memory defenses
- System Security Principles
  - 1. Protection vs. Security   2. Security Principles
    3. Mandatory Access Control
- Systems Security Mechanisms
  - 1. Multics   2. Linux   3. SELinux   4. eBPF
- Systems Security Problems
  - 1. Extensibility   2. Isolation 3. Software Security
    4. File Systems Security
- System Architectures
  - 1. Distributed Systems Security 2. Hardware Security

What Kind of Threats?

- What is the biggest concern?

Adversary-Controlled Code (Bad Code)

- Adversary may control the code that you run
- Examples
  - Classical: Viruses, Worms, Trojan horses, …
  - Modern: Client-side scripts, Macro-viruses, Email, Ransomware, Hijacking processes, …
- Easier to update/add software (malware) than ever

- What are the problems with adversary code on your machine?

Adversary-Controlled Code (Bad Code)

- You run an adversary-controlled program
  - What can an adversary do?

Adversary-Controlled Code (Bad Code)

- You run an adversary-controlled program
  - What can an adversary do?
  - Anything you can do
- Do you have anything you would want to protect?
  - Secret data on your computer
  - Communications you make with your computer
- Well, at least these are only "user" processes
  - They do not *directly* compromise the host
    - Beware "local exploits" that can "root" your system

Defenses

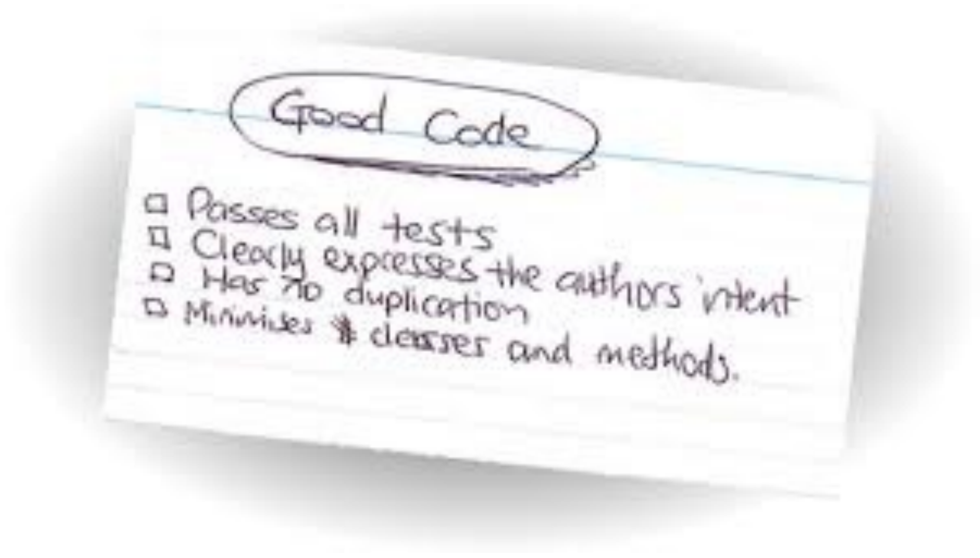
- What can you do to avoid executing adversary-controlled code?

Defenses

- What can you do to avoid executing adversary-controlled code?
- Defenses
  - Only run “approved” code
    - How do you know?
    - Use automated installers or predefined images
      - Let someone else manage it
  - “Supply chain security”

Good Code

- Fortunately, most code is not adversary controlled
  - I think…
- What is the problem with running code from benign sources?



Good Code

- Passes all tests
- Clearly expresses the authors intent
- Has no duplication
- Minimizes # classes and methods.

Good Code, May Go Bad

- Fortunately, most code is not adversary controlled
  - I think…
- What is the problem with running code from benign sources?
  - Not really designed to defend itself from a determined, active adversary

- Functions performed by benign code may be exploited – i.e., have vulnerabilities

Vulnerabilities

- A program vulnerability consists of three elements:
  - A flaw
  - Accessible to an adversary
  - Adversary has the capability to exploit the flaw
- Often focus on a subset of these elements
  - But all conditions must be present for a true vulnerability

Good Code – Goes Bad

- Classic flaw: Buffer overflow
- If adversary can access, exploits consist of two steps usually
  - (1) Gain control of execution – IP or stack pointer
  - (2) Choose code for performing exploitation
- Classic attack:
  - (1) Overwrite return address
  - (2) Write code onto stack and execute that

Good Code – Defenses

- Preventing either of these two steps prevents a vulnerability from being exploited
- How to prevent overwriting the return address?
  - ???
- How to prevent code injection onto the stack?
  - ???
- Are we done?
  - End the semester early…

Good Code – Evading Defenses

- Unfortunately, no
- (1) Adversaries gain access to the control flow in multiple ways
  - Function pointers, other variables, heap variables, etc.
  - Or evade defenses – e.g., disclosure attacks
- (2) Adversaries may perform desired operations without injecting code
  - Return-to-libc
  - Return-oriented attacks

Good Code – Confused Deputy

- And an adversary may accomplish her goals without any memory errors
  - Trick the program into performing the desired, malicious operations
- Example “confused deputy” attacks
  - SQL injection
  - File system attacks
  - Bypass attacks
  - Race condition attacks (TOCTTOU)

Good Code – Confused Deputy

- Classic flaw: SQL Injection
- If adversary can access, exploits consist of two steps usually
  - (1) Provide input used in query – employee id
  - (2) That is built into query code – name == eid
    - name == “x or x==x”
- Classic attack:
  - (1) Provide unsanitized input
  - (2) Used to generate code or perform an operation chosen by the attacker (data and code are mixed)

Who Defends These?

- Multiple parties can prevent attacks
  - Programmers
  - Compilers
  - Operating Systems
  - Hardware

Who Defends These?

- Multiple parties can prevent attacks
  - Programmers
  - Compilers
  - Operating Systems
  - Hardware
- Historically, the expectation is that the operating system would prevent all attacks
  - No matter how poor/malicious the code
  - With no help from compilers or hardware
- This didn’t work

Who Defends These?

- Multiple parties can prevent attacks
  - Programmers
  - Compilers
  - Operating Systems
  - Hardware
- Now, there are efforts underway at all levels to improve defenses
  - But, these efforts are still under development and ad hoc
  - Ultimately, the OS will need to coordinate defenses
    - E.g., apply most efficient use of defenses given the threats

Take Away

- In this class, we will focus on the methods to develop secure systems with a combination of techniques
  - Harder to distribute bad code
  - Harder to turn good code bad
  - Harder to leverage bad code for malicious purposes
- Difficult/expensive to prevent such problems entirely
  - Often applications perform unsafe actions
  - So, we cannot just block every action that could lead to an attack with blocking some necessary function
- We will need to trade-off function and security

Questions