CSE 544
Advanced Systems Security

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

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
  - OpenStack – Open source, IaaS cloud platform
  - Server and middleware – Web servers, browsers, window mgrs, system software…
- Office: W359 Westgate Bldg; Hours: W 1-2 and by appt
- Email: tjaeger@cse.psu.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

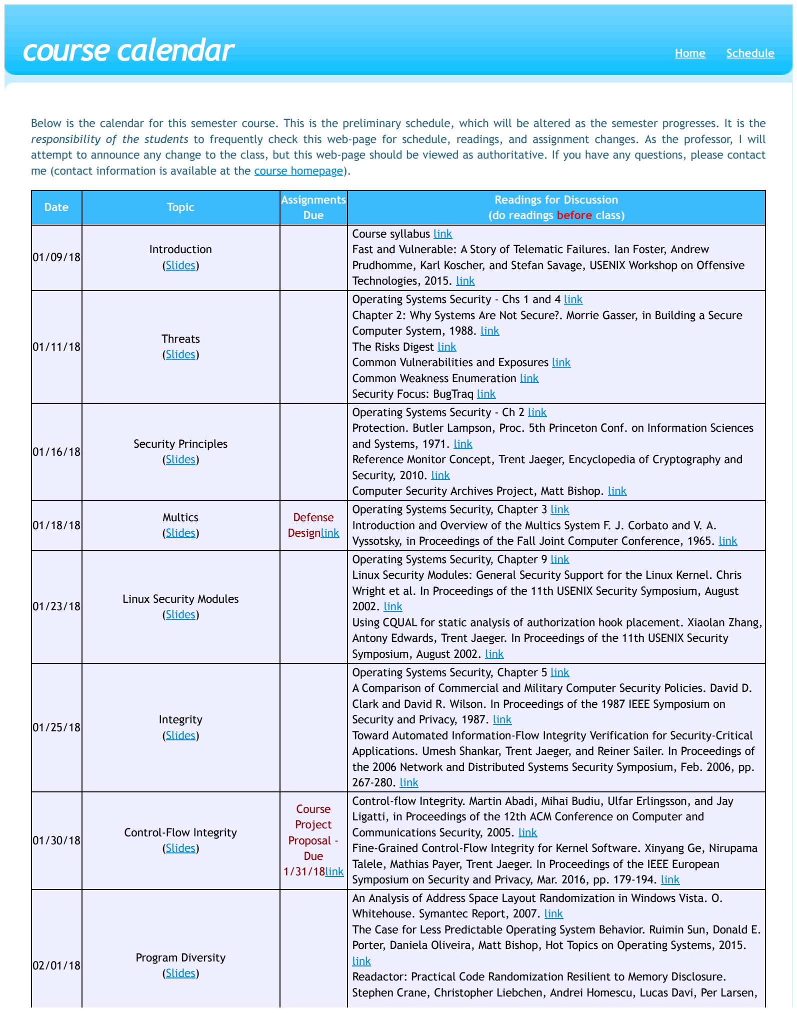
- Required:
  - CSE 543, CMPSC 458 (networks), CMPSC 411 (OS)
- Expected:
  - Solid OS and software background
- Additional:
  - Willingness to read
    - We are going to read a lot of systems security papers
  - Willingness to program
    - We are going to have an OS programming assignment (Linux) and systems course project

Course Materials

- Website
  - http://www.cse.psu.edu/~tjaeger/cse544-s18/
  - Course assignments, slides, etc. will be placed here
    - Check back often -- I may change some of the papers/assignments
- Course Textbook
  - My book: *Operating Systems Security*
    - *Available for free from inside PSU network – Google "Operating Systems Security, Trent Jaeger"*
  - Augmented with research papers

Course Calendar

- The course calendar has all the details
- Links to online papers for readings
- Links to projects
- 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,

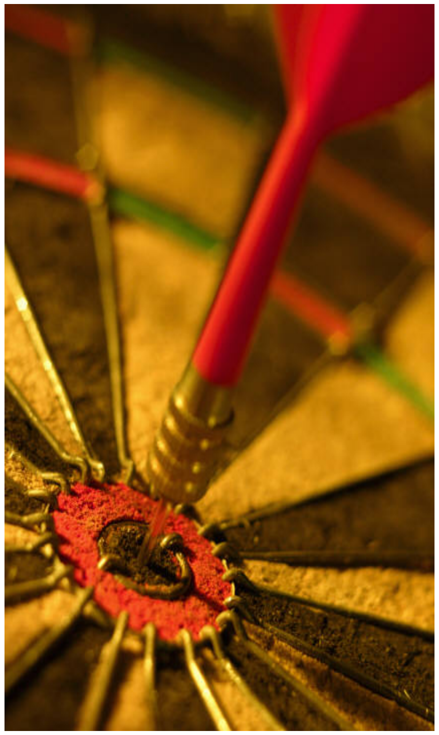
Course Mailing List

- Via Canvas
  - Use with care
- I will send a test email
  - Please reply if you do not receive by Fr
  - May need to forward to your CSE account
- Can use to email me
  - Please use “544” in the subject

Grading

PENNSTATE 1855

- Exams (55%)
  - Midterm (25%)
    - Take home
  - Final (30%)
    - In class
- Projects (35%)
  - Design and programming project
  - Course Project
- Participation (10%)
  - Be prepared with readings – possible quizzes



Lateness Policy

- Assignments and project milestones are assessed a 20% per-day late penalty, up to a maximum of 4 days. Unless the problem is apocalyptic, don't give me excuses. Students with legitimate reasons who contact the professor before the deadline may apply for an extension.
- You decide what you turn in

Academic Integrity

- See Computer Science and Engineering Department's Policy on Academic Integrity Standards
  - http://www.eecs.psu.edu/students/resources/EECS-CSE-Academic-Integrity.aspx

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. What is security? 2. Threats
- System Security Principles
  - 1. Protection vs. Security 2. Security Principles
- Systems Security Mechanisms
  - 1. Multics 2. Linux 3. SELinux
- Systems Security Problems
  - 1. Program Integrity 2. Confused Deputy 3. Confinement 4. Malware
- System Architectures
  - 1. Security Kernels 2. Capability Systems 3. VM Security
- Special Topics (Systems)
  - 1. New Hardware Features 2. Trustworthy Computing 3. Cloud Security
- Special Topics (Software)
  - 1. Information Flow Control 2. Symbolic Execution 3. Program Retrofitting

What Kind of Threats?

- Lead to security problems…
  - Consider XSS

Bad Code

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

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

Bad Code - Example

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

Bad Code - Example

- 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"

Bad Code - Defenses

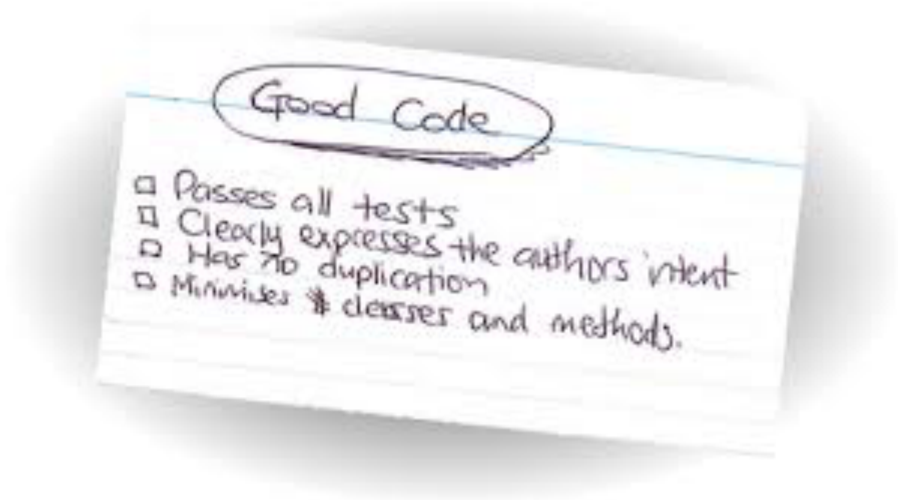
- What can you do to avoid executing adversary-controlled code?

Bad 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
  - "Sandbox" code you are uncertain of
    - How do you do that?

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

- 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

PENNSTATE 1855

- 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
  - Resource access attacks
  - Bypass attacks
  - Race condition attacks (TOCTTOU)

Result

- Adversaries have a variety of ways to try to get code under their control running on your computer
- Software defenses may not prevent exploitation
  - And still lots of room for improvement
- Malware and intrusion detection is a hard problem
  - How do we know whether code is bad or good?
- Systems security is about blocking damage or limiting damage from adversary-controlled execution
  - Not doing well enough yet

Who Has a Role?

- Who may be responsible for software and systems security in computing environments?
- Programmers (may be multiple groups)
- OS Distributors
- Administrators
- Users
- Service Providers
- Content Providers

Take Away

- In this class, we will focus on the methods to make the adversaries’ task more difficult
  - Harder to distribute bad code
  - Harder to turn good code bad
  - Harder to leverage code for malicious purposes
- Difficult to prevent such problems completely
  - 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