CMPSC 497

Special Topics: Software 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: Systems and Software 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: M 1-2 and W 2-3
• Email: tjaeger@cse.psu.edu
This course....

• Is a **programming** course that teaches techniques to avoid creating vulnerabilities and to add security mechanisms to protect your programs

• Caveat: We are still trying to figure out both

• Topics: What are … Program flaws and how to they become vulnerabilities? … Safe programming techniques to avoid vulnerabilities? … Tools and techniques to detect vulnerabilities? … Security mechanisms and how to add them to your programs?
Background

• Required:
  ‣ CMPSC 473, CMPSC 443

• Expected:
  ‣ Program in C

• Additional background:
  ‣ Programming Languages
    • We will learn some program analysis techniques to detect vulnerabilities and write secure code
Course Materials

• Website
  ‣ http://www.cse.psu.edu/~trj1/cmpsc497-s18/
  ‣ Course assignments, slides, etc. will be placed here
    • Check back often -- I may change some of the assignments

• Course Readings
  ‣ Papers available on the website
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

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).

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Assignments</th>
<th>Readings for Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/09</td>
<td>Introduction</td>
<td>Course syllabus</td>
<td></td>
</tr>
<tr>
<td>01/16</td>
<td>Passwords</td>
<td>Linux Password and Shadow File Format.</td>
<td></td>
</tr>
<tr>
<td>01/18</td>
<td>Programming Flaws (Info Flow)</td>
<td>Violent Injection Cheat Sheet and Tutorial.</td>
<td></td>
</tr>
<tr>
<td>01/23</td>
<td>Programming Flaws (Buffer Overflows)</td>
<td>Smashing the Stack for Fun and Profit. k. Thompson, Phrack 7(49), 1996.</td>
<td></td>
</tr>
<tr>
<td>02/01</td>
<td>Confused Deputy</td>
<td>The Confused Deputy (or why capabilities might have been invented). Norm Hardy, Operating Systems Review, pp. 36-38, Oct. 1988.</td>
<td></td>
</tr>
<tr>
<td>02/06</td>
<td>Defensive Programming</td>
<td>Secure Programming HOWTD (Chapter 1).</td>
<td></td>
</tr>
<tr>
<td>02/08</td>
<td>Defensive Programming</td>
<td>Secure Programming HOWTD (Chapter 5).</td>
<td></td>
</tr>
<tr>
<td>02/13</td>
<td>Penetration Testing</td>
<td>Restoration Tutorial.</td>
<td></td>
</tr>
<tr>
<td>02/15</td>
<td>Fuzz Testing</td>
<td>American Fuzzy Logic.</td>
<td></td>
</tr>
<tr>
<td>02/20</td>
<td>Static Analysis</td>
<td>TBD - Static Analysis Tutorial.</td>
<td></td>
</tr>
<tr>
<td>02/22</td>
<td>Static Analysis</td>
<td>TBD - Static Analysis Tutorial.</td>
<td></td>
</tr>
<tr>
<td>03/20</td>
<td>Execution Integrity</td>
<td>Control-Flow Integrity: Precision, Security, and Performance. Section 2.1.</td>
<td></td>
</tr>
<tr>
<td>03/22</td>
<td>Execution Integrity</td>
<td>Securing Software by Enhancing Data Flow Integrity. Section 2.1.</td>
<td></td>
</tr>
<tr>
<td>03/27</td>
<td>Comparing Java to C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 “CMPSC 497” in the subject
Grading

- Exams (60%)
  - Midterm (25%)
    - In class
  - Final (35%)

- Projects (30%)
  - Programming Projects
  - Homeworks

- 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. Today
- Software Vulnerabilities
  - 1. Information Flow
  - 2. Memory Errors
- Defensive Programming
  - 1. Safe Syscalls
  - 2. Attack Surfaces
  - 3. Design
- Finding Program Flaws
  - 1. Dynamic Testing
  - 2. Static Analysis
  - 3. Symbolic Execution
- Security Mechanisms
  - 1. Authorization
  - 2. Privilege Separation
  - 3. Execution Integrity
- Safe Programming Environments
  - 1. Memory Safe
  - 2. Information Flow-Safe
- Retrofitting Software for Security
  - 1. Authorization
  - 2. Privilege Separation
What Kind of Threats?

- Lead to security problems...
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 use an adversary-controlled library
  - What can an adversary do?
Bad Code - Example

- You use an adversary-controlled library
  - 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
  - But, they may 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?
  ‣ “Sandbox” code you are uncertain of
    • How do you do that?
  ‣ Use automated installers or predefined images
    • Let someone else manage it
Good Code

• You aim to write programs that does what you want

• What is the problem with running code from benign sources?
Good Code

• You aim to write programs that does what you want

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

• Pressure is on programmers to eliminate flaws that may lead to vulnerabilities (accessible and exploitable)
  ‣ Hard to identify program flaws
  ‣ Some program flaws are complex or expensive to prevent
  ‣ Programmers typically do not know where adversary access is possible
  ‣ Exploit methods are becoming more powerful

• On the positive side, a number of defenses are now known and some are practical
  ‣ How do programmers use them?
Take Away

• In this class, we will focus on the methods to write programs that make the adversaries’ task more difficult
  ‣ Harder to turn good code bad
  ‣ Harder to leverage code for malicious purposes

• We will look at several techniques
  ‣ How to write safe code
  ‣ How to detect flaws in code
  ‣ How to apply security mechanisms to code
  ‣ How should we add security to our programs