28 Lecture: Security/Wrapping Up

Outline:
- Announcements
- Final Exam
- Structure
- Wrapup
  - What are Operating Systems?
  - Major subject areas for this course
  - Why is this relevant to you?
- Official Class Photo
  - What do you want to talk about?
- And now for something completely different: Security
  - What it is
  - Where do vulnerabilities come from?
  - Real nature of the work: That’s funny
  - What to do
  - Demonstrations
  - Other useful topics
    - VM Tricks
- Floating Point

28.1 Announcements

- Coming attractions:

<table>
<thead>
<tr>
<th>Event</th>
<th>Subject</th>
<th>Due Date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>asgn5</td>
<td>minget and minls</td>
<td>Wed 5 23:59</td>
<td></td>
</tr>
<tr>
<td>asgn6</td>
<td>Yes, really</td>
<td>Fri 7 23:59</td>
<td></td>
</tr>
<tr>
<td>final</td>
<td>stuff</td>
<td>Sat 8 10:10</td>
<td>(in 03-201)</td>
</tr>
</tbody>
</table>

Use your own discretion with respect to timing/due dates.

- I’ll plan on being at at least the beginning of lab Friday.
- Finals Week Office Hours
  - Monday 10:00am–noon
- Remember to do asgn5 and asgn6 (two and five, so far).
- No class next time.
- Plans:
  - Answer questions
  - Class picture
- Note: Saturday’s final is not in the usual place.
- Don’t confuse memory and disk
28.2 Final Exam
28.2.1 Structure

Final: Chapters 1–5.

- short answer for breadth
- longer questions for depth

Emphasis on understanding: I might ask you what, but I’m more likely to ask why?, how?, or what if?.

28.3 Wrapup
28.3.1 What are Operating Systems?

resource allocation/performance/abstraction/reliability/tradeoffs

It all comes down to balancing:

1. Resource allocation
   - How to make the most of what you have
   - How to keep different processes from conflicting
   - How to make more than you have (faking it...)

2. Performance
   - Tuning the system for good performance while still remaining correct
   - scheduling, synchronization, resource allocation

3. Abstraction
   - Complexity management is key

4. Reliability
   - In the face of accident
   - In the face of malice

5. Tradeoffs
   *Faster, better, cheaper, choose two.*

Sounds like computer science, doesn’t it?

28.3.2 Major subject areas for this course

**OS Structure and History**  What are the issues w/OS. Why? What are they (and what are they not (almost everything))

**Processes**  What is a process, what do they need, how are they scheduled? What are the pitfalls (deadlock?)

**synchronization methods**
**Input/Output** How does an OS communicate with the outside world (and how do device drivers work)

**Memory Systems** How to solve the problem of multiple programs at once and not enough memory.
- Swapping,
- segmentation,
- virtual memory systems,
- paging algorithms.

**Filesystems** How to organize a large amount of persistent data:
- resource allocation
- performance
- consistency

**Security** Think about what’s going on.

28.3.3 Why is this relevant to you?
You’re not going to go out of here and write operating systems, so why study them:
- you will write large, concurrent systems
- all the principles here affect how programs behave and how they should be written
- consciousness of these issues will make you a better computer scientist; possibly a better human being. :)

28.4 Official Class Photo
In which this class is immortalized on the web.

28.4.1 What do you want to talk about?
28.5 And now for something completely different: Security

Security is not an add-on feature, it must be built in. It is a state of mind and a design principle. (In fact, we’ve been talking about it all along.)

28.6 What it is

...and nothing else.

We’re really talking about Information Assurance.

- Requires looking at things differently. A thing may be exactly what it seems...and something else.
  - buffer overflow
  - drawer of old souveniers, etc.

- Computer security is System security (Bad guys don’t have to play by the rules.) Ex:
  - The Gordian Knot
  - Indiana Jones and the pistol
  - Fights in New Jersey?
  - the Maginot line
  - stealing a whole ATM
  - Manufacture of RFID passports outsourced

Principle of Easiest Penetration (Pfleeger)

An intruder must be expected to use any available means of penetration. The penetration may not necessarily be by the most obvious means, nor is it necessarily the one against which the most solid defense has been installed.

- Unfortunately, the good guys do have to play by the rules (this is harder)
- The threats are real (though I shouldn’t have to tell y’all this)
  - 11/08 Acroread vulnerability: Javascript buffer overflow
  - New RFID Passports?
    Contains in digital form (RFID):
    * date of birth,
    * gender,
    * birthplace
    * issuance and expiration dates
    * photograph
    Outsourced to Thailand?

- There isn’t a hacker behind every rock and tree, but there is behind every other rock and tree.
• By the way, what are we protecting?

Think in terms of three security Services:

<table>
<thead>
<tr>
<th>confidentiality</th>
<th>will those things that are supposed to be kept secret in fact be kept secret?</th>
</tr>
</thead>
<tbody>
<tr>
<td>integrity</td>
<td>Can we rely on the validity of the information provided.</td>
</tr>
<tr>
<td>availability</td>
<td>Will we be able to get to it when we need it.</td>
</tr>
</tbody>
</table>

One must think not in terms of how things are supposed to be used, but how they can fail. Consider:

 [...] Security engineering involves making things not happen. It involves figuring out how things fail, and then preventing those failures.

In many ways this is similar to safety engineering. Safety is another engineering requirement that isn’t simply a “feature.” But safety engineering involves making sure things do not fail in the presence of random faults: it’s about programming Murphy’s computer, if you will. Security engineering involves making sure things do not fail in the presence of an intelligent and malicious adversary who forces faults at precisely the worst time and in precisely the worst way. Security engineering involves programming Satan’s computer.

And Satan’s computer is hard to test.


Complete essay available at: http://www.counterpane.com/crypto-gram-9911.html#WhyComputersareInsecure

So, with regard regard to each of CIA, we want to look at:

• Threats circumstances that have the potential to cause harm (“badness”)
  This is intentionally left vague: value will vary from system to system and environment.

• Vulnerabilities are weaknesses in the system that could be exploited to cause loss or harm

• Controls (or countermeasures) are actions, devices, techniques, etc., that remove or reduce a vulnerability.
  Controls need to be proportional. It’s only necessary to mitigate the vulnerability to the point where it’s not worth bothering. Anything else is wasted. (e.g. Greg’s bike)

Remember, systems are vulnerable where they are weak (not where we expect attacks).

Principle of Easiest Penetration (Pfleeger)
An intruder must be expected to use any available means of penetration. The penetration may not necessarily be by the most obvious means, nor is it necessarily the one against which the most solid defense has been installed.

Example:

• xkcd example
• Small: U.C.D.
• Large: Maginot line
28.7 Where do vulnerabilities come from?

- Careless programmers hall of shame: the dumbest mistake in the world, buffer overflow.
  - `gets()`
  - `strcat()`
  - `strcpy()`
  - `scanf()`

Don’t let this happen to you.
Moral: Don’t ever assume anything about the data you are receiving or the environment.

- Careless sysadmins/designers
  - writeable `/etc/passwd`
  - writeable root directory

- Poor/sloppy design
  - Bad design: Dumb VM tricks: TENEX password flaw.
    Combine:
    * Password protection on individual files.
    * User-visible paging statistics.
    * Short-circuit password evaluation.
  - old (possibly apocryphal) vulnerability in `su`

  **The setup** `su` assumed that if it was unable to open the passwd file, that the system was experiencing a catastrophic failure and allowed root access.

  **The exploit** open random files until the file descriptor table fills up and `exec()` `su`...

  - `mkdir`:
    `mkdir` used to be a `mknod` followed by a `chown`. On a slow system there would be a race condition that would allow a user to capture ownership of any file:

    | `mkdir foo` | `attack` |
    |-------------|---------|
    | `mknod foo` | `rm foo` |
    |             | `ln -s /etc/passwd foo` |
    | `chown user foo` | `root mail exploit` |

- Then there are dumb users:
  - my passwd is “pizza”
  - it’s on the bulletin board
  - it’s always logged in
  - social engineering.

Remember, systems are vulnerable where they are weak (not where we expect attacks).
28.8 Real nature of the work: That’s funny

- sonic attack?

28.9 What to do

- **Principle of Least Privilege** Don’t ever use more privilege for any operation that is necessary.
  
  Consider role-based mechanisms vs. only root/user

- **Check parameters and enforce boundaries** Make sure that failures cannot propagate.
  
  -- Internal: Consider the tty driver’s behavior:

  ```c
  if (umap(m_ptr->PROC_NR, (vir_bytes)m_ptr->ADDRESS, m_ptr->COUNT) == 0) { 
    r = EFAULT;
  } else { ... 
  }
  ```

  -- External: See Fig 58
  
  - If I gave you a gun and told you it was unloaded, would you believe me?
  
  - Check return values: Matt B.’s example

- **Do something reasonable** (Principle of least astonishment)

- **Clear memory pages/disk blocks** Information is persistent even when “undefined”.

- **Consider Security/Usability Tradeoffs** (If a PC is buried in concrete, it’s more secure, but decidedly less useful.)

28.10 Demonstrations

(Postponed due to uncooperative demo platform.)

- Stack smashing

- %n conversion
XSS?

Principles:

<table>
<thead>
<tr>
<th></th>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Principle of Least Privilege (f)</td>
<td>Every program and every user of the system should operate using the least set of privileges necessary to complete the job.</td>
</tr>
<tr>
<td>2.</td>
<td>Principle of Economy of Mechanism (a)</td>
<td>Keep the design as simple and small as possible.</td>
</tr>
<tr>
<td>3.</td>
<td>Principle of Open Design (d)</td>
<td>Design should not be secret. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords.</td>
</tr>
<tr>
<td>4.</td>
<td>Principle of Complete Mediation (c)</td>
<td>Every access to every object must be checked for authority. Both direct access, and attempts to circumvent access control should be considered.</td>
</tr>
<tr>
<td>5.</td>
<td>Principle of Fail-Safe Defaults (b)</td>
<td>Base access decisions on permission rather than exclusion.</td>
</tr>
<tr>
<td>6.</td>
<td>Principle of Separation of Privilege (e)</td>
<td>When feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.</td>
</tr>
<tr>
<td>7.</td>
<td>Principle of Least Common Mechanism (g)</td>
<td>Minimize the amount of mechanism common to more than one user and depended on by all users.</td>
</tr>
<tr>
<td>8.</td>
<td>Principle of Psychological Acceptability (h)</td>
<td>Pfleeger: “Ease of Use”) It is essential the the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms.</td>
</tr>
</tbody>
</table>