# Course Outline

CS3235—Computer Security
Sandeep Kumar
skumar@comp.nus.edu.sg (Use subject: CS3235 . . . . . .)

**Textbook**: Security in Computing by Charles Pfleeger $2^{nd}$ ed.
**Office Hours**: TBD.
**Course Web Page**: `http://www.comp.nus.edu.sg/~cs3235/`

- Introduction (Chapter 1): 1 lecture.
- Historical ciphers and their (brief) cryptanalyses (Chapter 2): 2 lectures.
- Basic block ciphers, Feistel networks and DES, Elementary number theory, RSA, DH, El-Gamal, Hashing, MACs (Chapter 3): 3 lectures.
- Protocols (Chapter 4): 1 lecture.
- Viruses and other malicious code (Chapter 5): 1 lecture.
- Protection in Operating Systems (Chapter 6): 2 lectures.
- Database Security (Chapter 8): 1 lecture.
- Network Security (Chapter 9): 2 lectures.
- Information flow, Anonymity, sundry topics (time remaining).

# Grading

This is tentative.

10% Tutorials.
20% Project(s): TBD. Probably implementing DES.
30% Midterm.
40% Final.

## Why is Computer Security Necessary?

Adapted from [Kan01].

- Because a lot of money is handled by computers.
- Because a lot of important information is stored on and handled by computers.
  - Would you want anyone to find your GPA, SAT, GRE scores?
  - How about your credit history, your medical history?

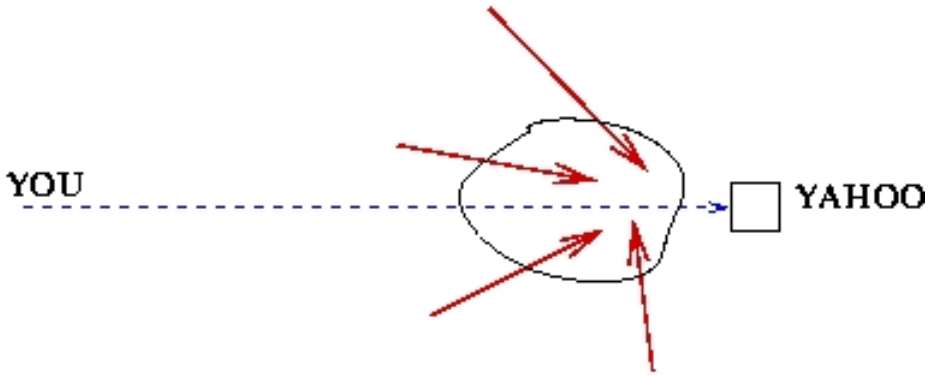
  There needs to be a mechanism to control sharing of information.
- Because society is increasingly dependent on the correct operation of computers. See http://www.zdnet.com/anchordesk/stories/story/0,10-738,2873733,00.html.

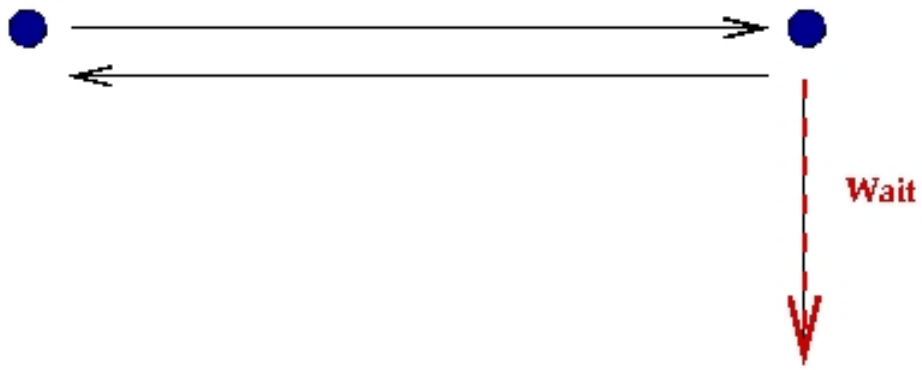
## Examples of Security Problems

Adapted from [Kan01].

- The Internet Worm c1988 (buffer overflow).
  - Spread over the Internet to many sites.
  - Around 6000 sites were shut down to get rid of it.
- Virus Attacks.
- Denial of Service Attacks.
  - Flooding of web servers with enormous # of requests.
  - Flooding networks enroute to the target.
  - Exploiting target TCP state machines.

# Examples of Denial of Service



## Information Security

Traditional elements of information security—termed *goals* in [Pfl96].

- Confidentiality [they want your data]. *Assets of a computing system are accessible only to authorized parties. Includes reading, printing, or even testing for existence of an object.*
  Breach: interception.
- Integrity. *Assets can be modified only by authorized parties. In security, usually distinct from structural integrity (well formedness).*
  Breach: modification, fabrication.
- Availability [they want your bandwidth, cpu, disk]. *Assets are accessible to authorized parties.*
  Breach: interruption.

## Why is security hard?

Adapted from [Kan01].

- Tradeoff between convenience and security, or performance and security.
- Wily human opponents seek to outwit us. Must assume that the opponent will attack the weakest point.
- Must get everything right–any mistake is an opportunity for the opponent.
- Bug-free software?

## Design Principles for Secure Systems

Saltzer and Schroeder [SS75].

1. Economy of [protection] mechanism. Keep the design as simple and small as possible.
2. Fail-safe defaults. Base access decisions on permission rather than exclusion.
3. Complete mediation. Every access to every object must be checked for authorization.
4. Open design. Security through obscurity is not.
5. Separation of privileges. Two locks are better than one!
6. Least privilege. Operate using the least privileges necessary to complete the job.
7. Least common mechanism. Minimize the amount of mechanism common to more than one user and depended on by all users.
8. Acceptability. Human interface should be easy to use.

---

**Lessons**: (Blaine Burnham)

- Security is not an add on.
- Assurance is important.
- It takes a secret to keep a secret i.e., key management is really hard.
- Security is not a silver bullet.
- Security is a system property.

## Buffer Overflow – Attack of the Decade

Adapted from [Bon] and [CWP+99].

- Extremely common bug
  - 1997: 16/28 CERT advisories.
  - 1998: 9/13 CERT advisories.
  - 1999: 6/12 CERT advisories.
- Often leads to total compromise of host.
- Requires expertise and patience (until someone posts an exploit).
- Two steps:
  - Inject suitable code in the program's address space.
  - Get the program to jump to that code, with suitable parameters loaded into registers and memory.

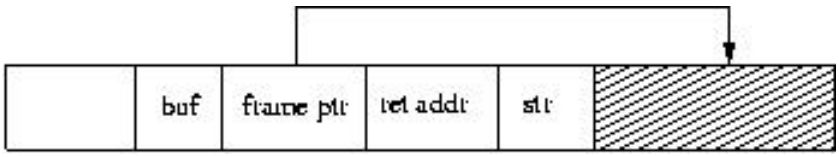
# Injecting code on the Activation Record

Suppose a web server contains the function:

```
void func(char *str)
{
  char buf[128];

  strcpy(buf, str);
  do_something(buf);
}
```

When the function is invoked, the stack looks like:

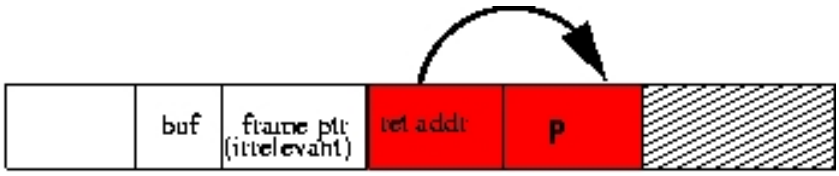



What if *str is $\geq 136$ bytes? After strcpy, the stack looks like:


The basic problem is that **strcpy** doesn't do range checking.

# Stack Smashing Attack

What if the buffer overflow results in the following stack state:




P: execve("/bin/bash", argv, envp) properly constructed!

A decompilation of the **libc** execl call expands to something like:

```
08059240 <__execve>:
 8059240:       55                      push   %ebp
 8059241:       b8 00 00 00 00          mov    $0x0,%eax
 8059246:       89 e5                   mov    %esp,%ebp
 8059248:       85 c0                   test   %eax,%eax
 805924a:       57                      push   %edi
 805924b:       53                      push   %ebx
 805924c:       8b 7d 08                mov    0x8(%ebp),%edi
 805924f:       74 05                   je     8059256 <__execve+0x16>
 8059251:       e8 aa 6d fa f7          call   0 <_init-0x80480b4>
 8059256:       8b 4d 0c                mov    0xc(%ebp),%ecx
 8059259:       8b 55 10                mov    0x10(%ebp),%edx
 805925c:       53                      push   %ebx
 805925d:       89 fb                   mov    %edi,%ebx
 805925f:       b8 0b 00 00 00          mov    $0xb,%eax
 8059264:       cd 80                   int    $0x80
```

- When **func()** returns, **/bin/sh** will read and write file descriptors $0$ and $1$.
- Attack code runs *on the stack*.
- Unsafe **libc** calls: strcpy, strcat, gets — no range checking.

## Exploiting buffer overflows

- If the web server calls **func**() with given URL, then an attacker can create a 200 byte URL to obtain shell on the web server!
- Some complications:
  - Program **P** shouldn't contain the `'\0'` character.
  - Overflow shouldn't crash the program before func() returns.
- Recent buffer overflows of this type:
  - Overflow in the MIME type field in MS Outlook.
  - Overflow in ISAPI in IIS.

# References


[Bon] Dan Boneh. CS155 lecture notes. Computer Science Department, Stanford University.

[CWP+99] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. DARPA Information Survivability Conference and Expo (DISCEX), 1999. Also published as an invited talk at SANS 2000.

[Kan01] Mohan Kankanhalli. CS3235 lecture notes. Computer Science Department, National University of Singapore, Fall 2001.

[Pfl96] Charles P. Pfleeger. *Security in Computing*. Prentice Hall, iind edition, 1996.

[SS75] Jerry Saltzer and M. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, September 1975.