CIS 520, *Operating Systems Concepts*

Lecture 11

Security

System Security

- In general, in this course, we are not dealing with network security—only with security of a single system (but these subjects are coupled!)
- For that matter, we primarily deal with the file system security, but we will learn *signatures* and *certificates* for application data exchange
- Two important facets: *data loss* and *intrusion*

Goals and Threats

Goal	Threat
1 Data confidentiality	Exposure of data
2 Data integrity	Tampering with data 篡改
3 System availability	Denial of service

记住

Data Loss

- Acts of God
- Hardware (or software) errors
- Human errors

How to deal with it?

- Maintain backup copies far away from the original data

Intruders

- Non-technical users prying
- Well-informed technical users snooping for fun
- Criminals attempting to make money (change software to truncate rather than round interest—remember how we were truncating random numbers?; encrypt the disk data and demand ransom)
- Commercial spies breaking into research databases
- State-serving spies breaking into military databases

Security Flaws—General Issues

- These have been found in virtually all operating systems
- Some resulted from the design, in which no one even expected the system to be attacked
- Others could not be foreseen because of the unpredictability of human behavior –overall, whatever humans use for their good, they also use for their destruction
- At the moment, armies of "gray hackers," computer scientists, and U.S. Government officials are trying to deal with the problem
- We will briefly go over some examples, but—remember—these are only *examples*. So far, it has been impossible to predict potential security problems.

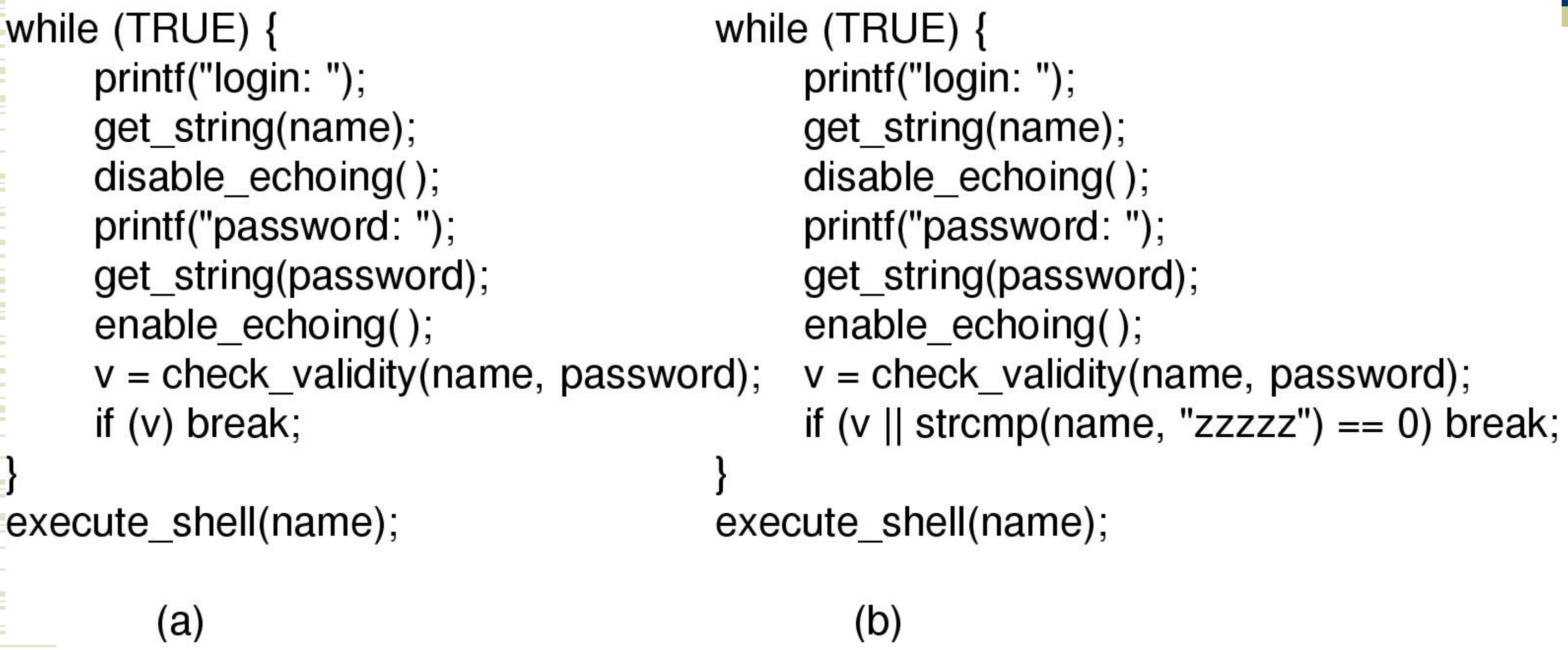
Examples of Security Flaws

- The *lpr* command, in earlier versions of *UNIX*, had an option to remove a file after it was printed. The `/etc/passwd` file could thus be printed (passwords are encrypted) and then removed
- One could link the *core* file (containing the program dump after it has crashed) with the `/etc/passwd` file, then crash a program, and thus force the file be overwritten
- The *mkdir* program used to be owned by the *root*. When I executed it (say, *mkdir my_dir)*, it first created the i-node for the directory *my_dir* using the *mknode* system call, and then changed the ownership of *my_dir* from the *root* user id to mine, using *chown* system call. It was possible then to insert a few instructions between these two calls to remove the i-node and link the password file to my directory under the name *my_dir*

Attacks from Inside

- *Trojan Horse* (a trickster program that has the same name as a well-known one)
- *Login spoofing*
- *Logic bomb* (an insider’s vengeance)
- *Trap door* (this is why code reviews are essential!)
- *Buffer overflow*

Trap Doors (A. Tanenbaum)



```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing( );
        printf("password: ");
        get_string(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v) break;
}
execute_shell(name);
```

(a)

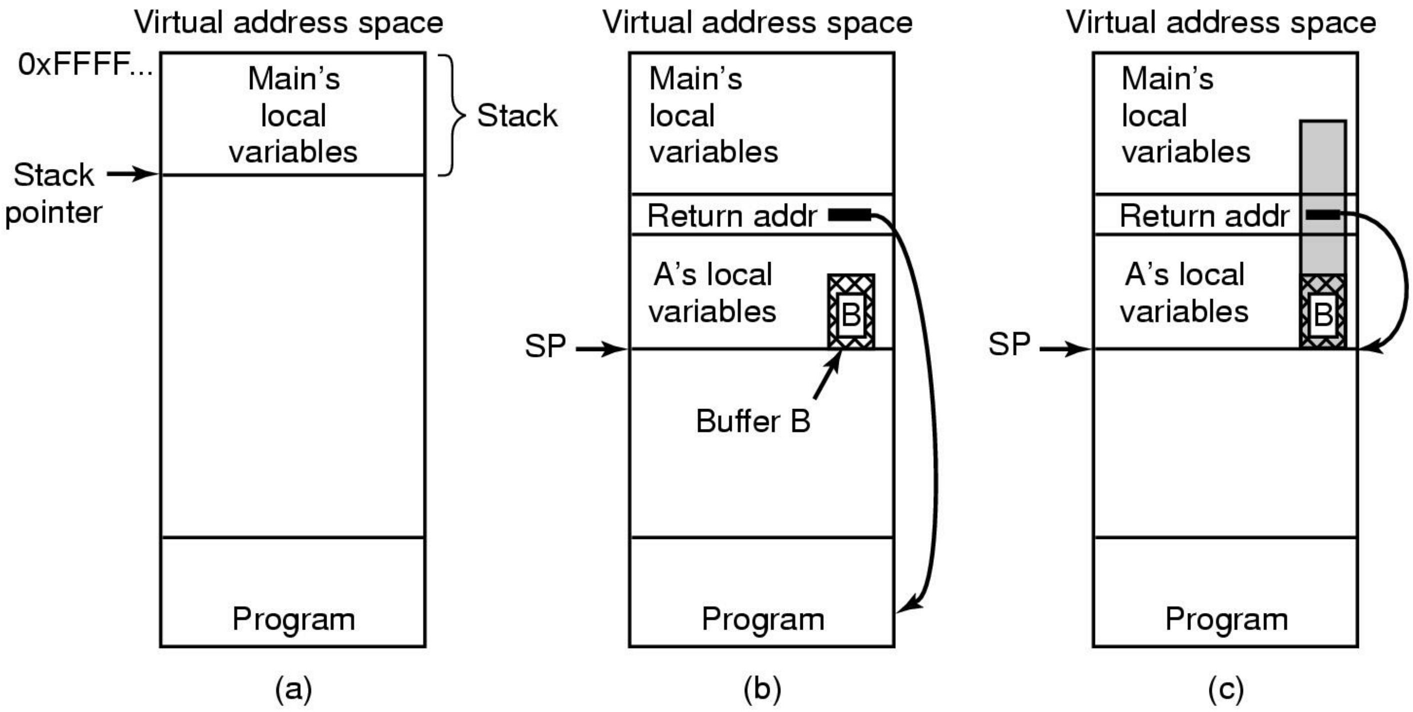
```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing( );
        printf("password: ");
        get_string(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v || strcmp(name, "zzzzz") == 0) break;
}
execute_shell(name);
```

(b)

(a) Normal code.

(b) Code with a trapdoor inserted

Buffer Overflow (A. Tanenbaum)

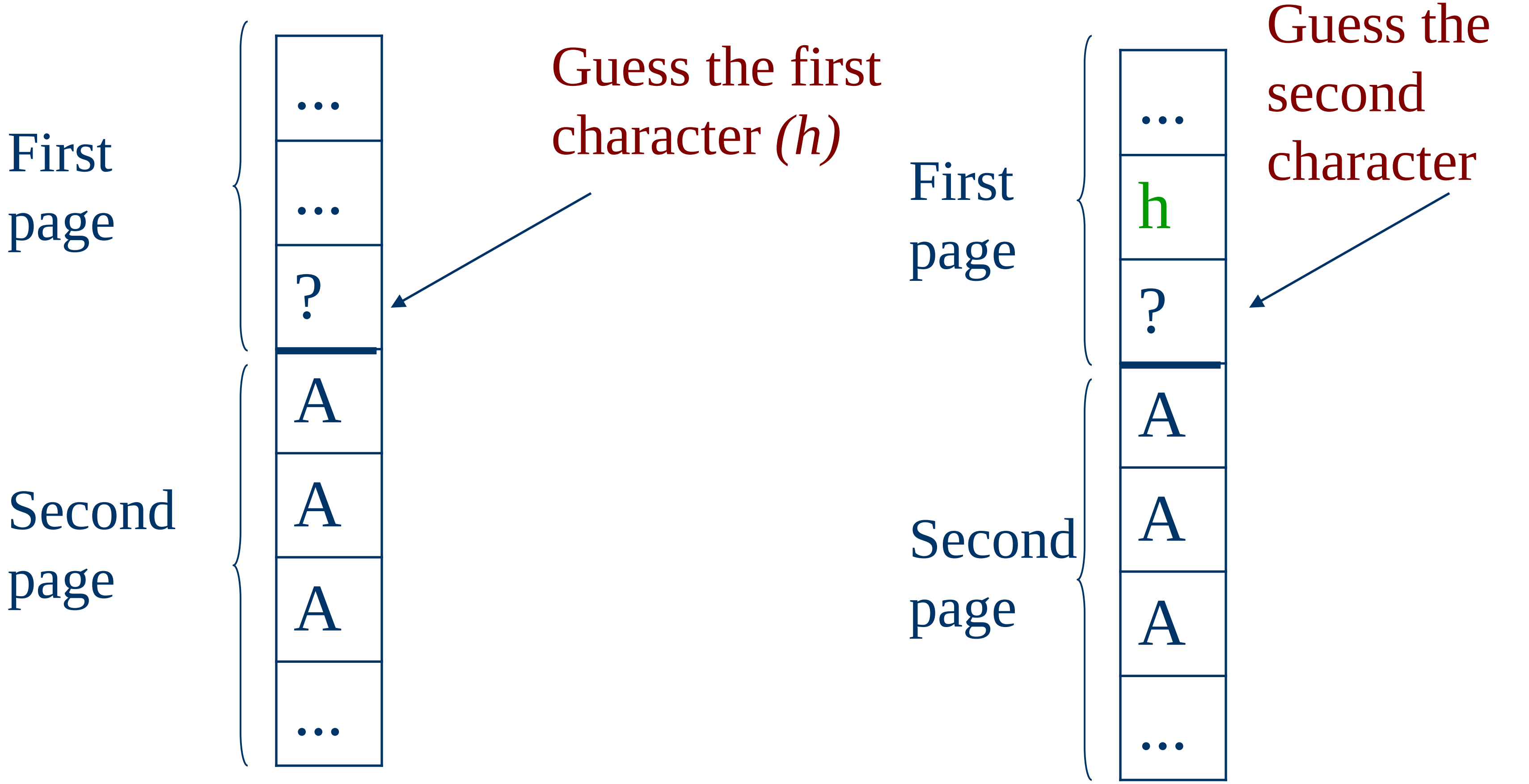


- (a) Situation when main program is running
- (b) After program *A* called
- (c) Buffer overflow shown in gray

Examples of Security Flaws (cont.)

- Now, see what happened with *TENEX* (most popular system on DEC-10 minicomputers). With *TENEX* a user program could catch a page fault interrupt
- The system password-protected each file. It checked a password one character at a time, and rejected it the moment the character was wrong
- An intruder could put the first character of the password as the last character of a page and ensure that the next page was forced out of memory (how?)
- When the system checked the password, an intruder would receive the page fault interrupt *only* if the first character was right (otherwise, the system would produce an error, and the intruder would try replacing the character—at most 120 times)
- As the result, an *n*-character password could be guessed with *120n* attempts (rather than 120^n attempts)

Examples of Security Flaws (cont.)



First page

Second page

...

...

?

A

A

A

...

Guess the first character *(h)*

First page

Second page

...

h

?

A

A

A

...

Guess the second character

Examples of Security Flaws (cont.)

- In OS-360, it was possible to start a tape read and then continue with another system call
- An intruder could start a tape read, and then issue an *open file* call with the intruder's own file and its password
- The system read the file structure and checked the password—that is when the tape read *overwrote* the name in the file structure with the name of the file the intruder was interested in. Before opening the file, the system read its name again (instead of saving it in the first place!)—and opened it!

Examples of Security Flaws (The Internet Worm)

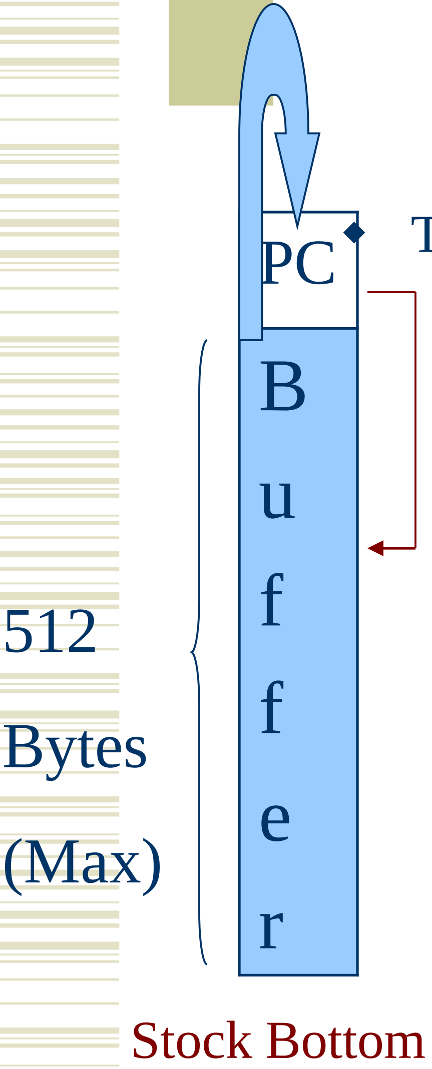
- *BSD UNIX* has a useful program, *finger*. A while ago, if you typed *finger* *igorf@arch3.att.com*, you would get a response that looked like that:

```
igorf logged on since 1:15 PM, April 11, 1994
In real life: Igor Faynberg
Idle time: 40 sec
Telephone: (908) 949-0137
Organization: Bell Laboratories Architecture Area
Mail last read @ 1.10 PM
No plan
```

Examples of Security Flaws (The Internet Worm)

- The *finger* is executed by the *finger daemon*, which runs in the background. When it gets a *finger* request (which can come from outside), it executes it as a procedure call—with the parameter on stack—and then returns to its *main()* program, which is a tight loop
- On November 2, 1988, Robert Tappan Morris (then a Cornell graduate student), a son of a famous security expert who worked in Bell Labs and then moved to the National Security Agency, wrote a 99-line *C* program called *l1.c*

Examples of Security Flaws (The Internet worm)



The worm did the following:

- It used *rsh* or *sendmail* to distribute itself to all the machines connected to the machine it was being executed (we will study later how this is done)
- If it failed, it would issue *finger* with a specially built 536-byte string as a parameter. The bulk of the request was VAX machine code that asked the system to execute the command interpreter *sh*, and the extra 24 bytes represent just enough data to write over the server's stack frame for the *main()* routine. When the *gets()* routine exited, it jumped to the worm’s code address the worm wrote over the program counter.
- Then it would try to *cryptanalize* the password file to gain any machine other users had accounts on
- Then it would continue distributing itself, except when it entered a machine on which it was already present. In this case it would exit *six* out of *seven* times (but leave another copy once in seven times)

Remember the Memory Reference String Example?

Page size: 250 *Instruction size*: 4 bytes (one word)

PC at: 480 *SP* at: 25004

480: *MOVEW @2000, R1*	*1, 7,*
484: *MOVEW R1, @SP*	
488: *SUBI SP, #4*	*1,*
492: *JSR @5120*	*1, 20, 100, [subroutine string], …*
496: *ADDI SP, #4*	
500: *CMP R0, #0*	*2,*
504: *JNZ @5152*	*2 or [2, 20]*

1, 7, 1, 100, 1, 1, 20, 100, ... 1, 2, 2

Examples of Security Flaws (The Internet worm)

- One out of seven was too many—thousands of machines were infested with the worm, and they effectively stopped
- A friend of Morris's spoke to the *New York Times*, and inadvertently gave the reporter, John Markoff, the login—*rtm*. This is how Morris was found
- He was evicted from the school, tried and convicted convicted of violating the computer Fraud and Abuse Act (Title 18), and sentenced to three years of probation, 400 hours of community service, a fine of $10,050, and the costs of his supervision.
- (Three things are unknown: his intensions in respect to the *worm*, his legal fees, and the financial damage to the industry. Some sources speculate that they exceeded $150,000. The financial loss caused by the worm is estimated between $10,000 and $10,000,000)
- Robert Morris is a professor at MIT now

Generic Security Attacks (all of them worked at least once!)

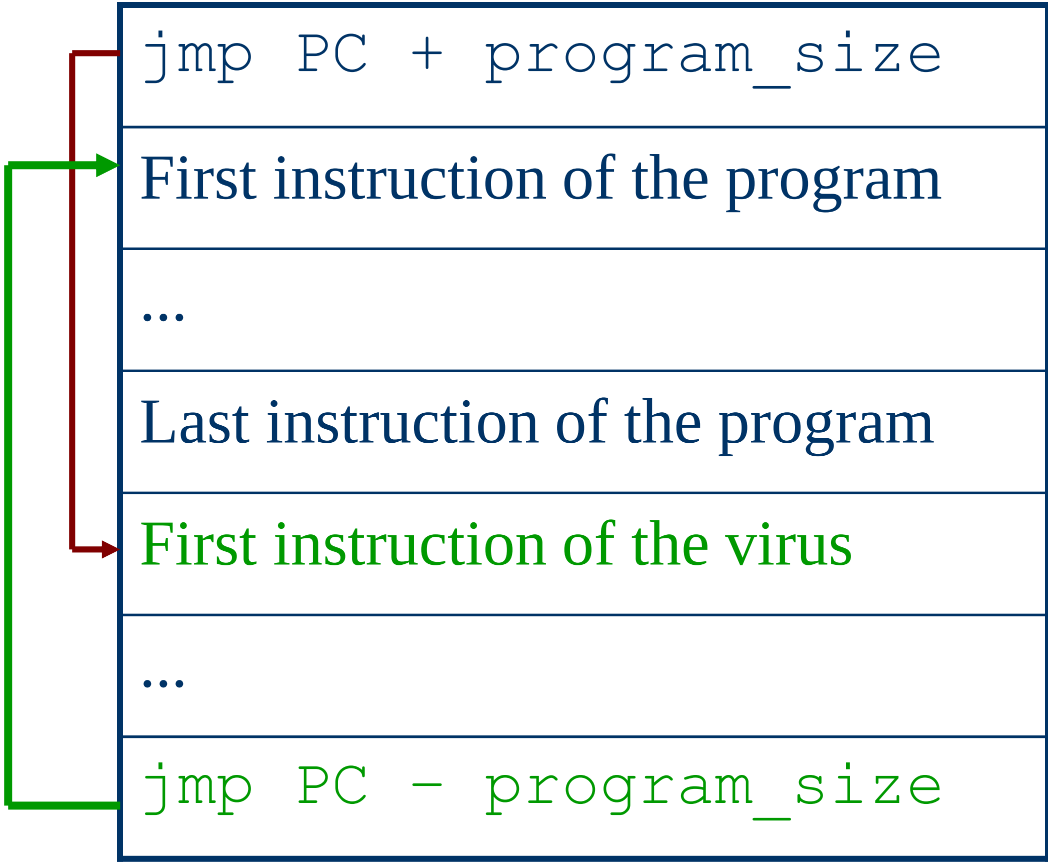
- Request memory pages, free disc space, or old tapes, and read them—there could be a lot of interesting information!
- Try illegal system calls, or legal calls with illegal (or just unreasonable) parameters—some systems may get confused enough to let you do what you want
- Try to modify the file system structure supplied with the *open* call. Changing it will often stun the system, too
- When asked for a password (on a network machine or a mainframe), hit *del* or *break*, or just type too much—the password checking program may... crash!

Generic Security Attacks (cont.)

- When someone left a computer terminal unattended, quickly write and executed a program that would prompt him or her for the login and password, mail these to you upon reception, and then execute *.profile* or otherwise pretend there is a normal start-up
- Carefully read the system manual. If it says “Never do <this thing>,” do this thing as many times as possible
- Befriend (or, better yet, charm—or even bribe) a systems administrator and try to obtain a *trapdoor*—absence of security checks for your login

Viruses

- A [computer] *virus* is a program fragment attached to a legitimate program. It differs from a worm in that the worm is a stand-alone program, but a virus is not
- When the infected program is started up, it immediately checks all other programs. If an uninfected program is found, the virus is attached to it, and the instruction is added in the beginning of its code to jump to the virus. The last instruction of the virus jumps to the second (previously, first) instruction of the program



```
jmp PC + program_size
```

First instruction of the program

…

Last instruction of the program

First instruction of the virus

…

```
jmp PC – program_size
```

What Can a Virus Do?
(in addition to infecting other programs)

- Erase or modify files
- Encrypt all files and leave a message on the screen to send $500 in crisp bills to a post office box in a remote country for a decryption key
- Infect the disk's boot sector and make it impossible to boot the computer (now, many people would pay $500 to be able to boot it again!)

Virus Damage Goals

- Blackmail
- Deny service for as long as the virus runs
- Permanently damage hardware
- Target a competitor's computer
  - do harm
  - spy
- Play inter-corporate dirty tricks
  - sabotage another corporate officer's files

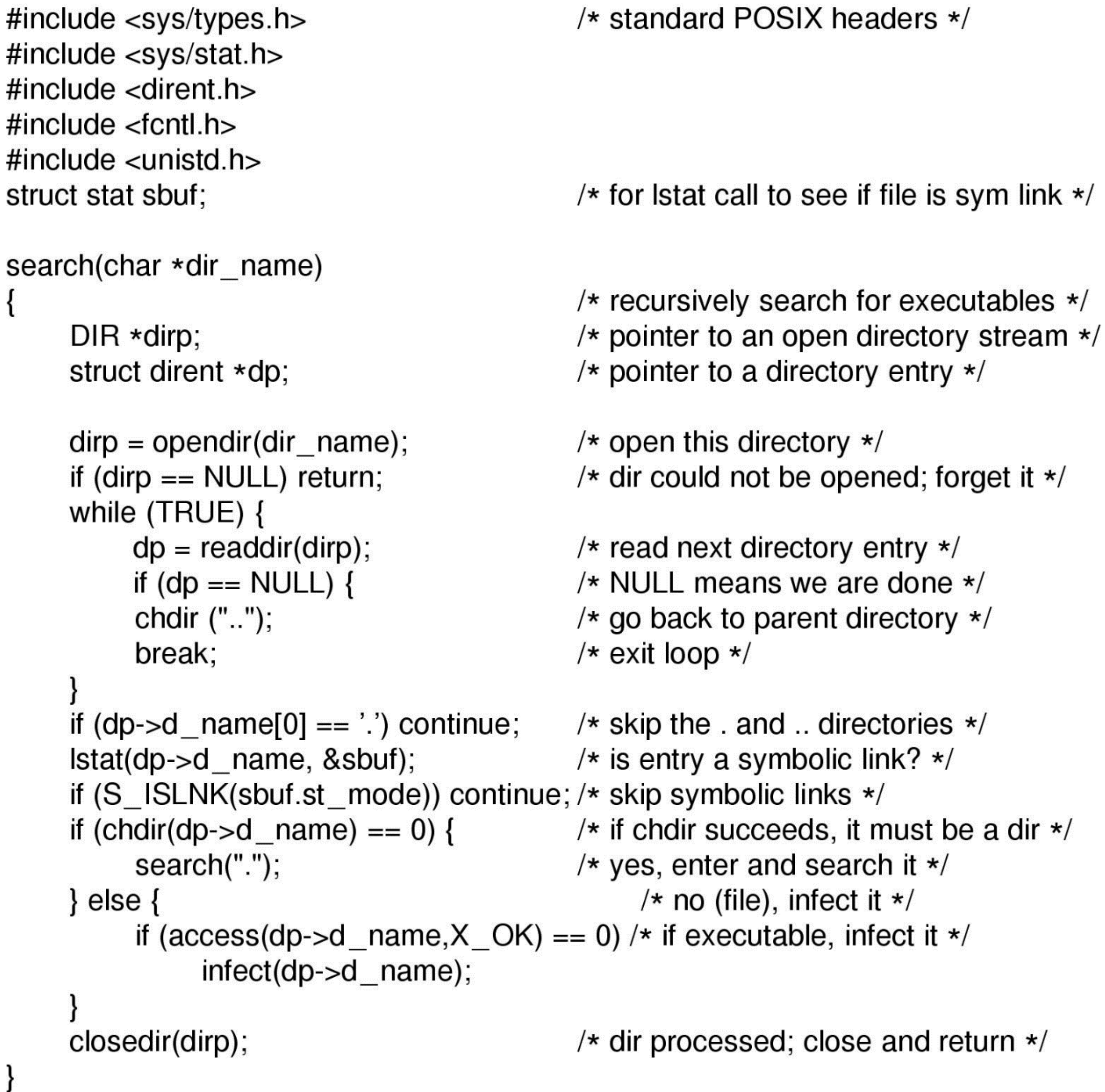
How Viruses Work (1)

- Virus is written in machine language
- Virus is inserted into another program
  - using a tool called a “dropper”
- Virus is dormant until the program is executed
  - then it infects other programs, and
  - eventually executes its “payload”

An example (A. Tannebaum)

Recursive procedure that finds executable files on a UNIX system

A virus can infect them all!

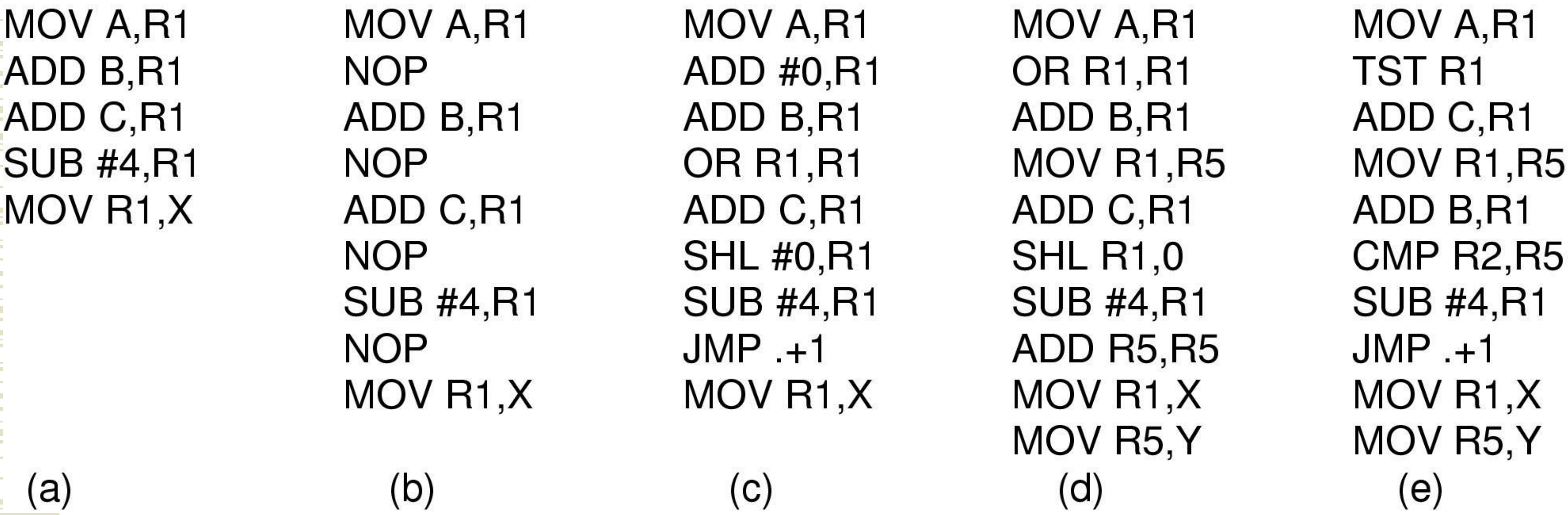


```
#include <sys/types.h>                          /* standard POSIX headers */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                               /* for lstat call to see if file is sym link */

search(char *dir_name)
{                                               /* recursively search for executables */
    DIR *dirp;                                  /* pointer to an open directory stream */
    struct dirent *dp;                          /* pointer to a directory entry */

    dirp = opendir(dir_name);                   /* open this directory */
    if (dirp == NULL) return;                   /* dir could not be opened; forget it */
    while (TRUE) {
        dp = readdir(dirp);                     /* read next directory entry */
        if (dp == NULL) {                       /* NULL means we are done */
        chdir ("..");                           /* go back to parent directory */
        break;                                  /* exit loop */
    }
    if (dp->d_name[0] == '.') continue;         /* skip the . and .. directories */
    lstat(dp->d_name, &sbuf);                   /* is entry a symbolic link? */
    if (S_ISLNK(sbuf.st_mode)) continue;        /* skip symbolic links */
    if (chdir(dp->d_name) == 0) {               /* if chdir succeeds, it must be a dir */
        search(".");                            /* yes, enter and search it */
    } else {                                        /* no (file), infect it */
        if (access(dp->d_name,X_OK) == 0) /* if executable, infect it */
            infect(dp->d_name);
    }
    closedir(dirp);                             /* dir processed; close and return */
}
```

How Viruses Spread

- Virus is placed where it is likely to be copied
- When copied, it
  - infects programs on the hard drive, memory stick, etc.
  - may try to spread over LAN
- Virus may be attached to innocent looking email
  - when it runs, it uses the address book to replicate

Polymorphic Code



```
MOV A,R1        MOV A,R1        MOV A,R1        MOV A,R1        MOV A,R1
ADD B,R1        NOP             ADD #0,R1       OR R1,R1        TST R1
ADD C,R1        ADD B,R1        ADD B,R1        ADD B,R1        ADD C,R1
SUB #4,R1       NOP             OR R1,R1        MOV R1,R5       MOV R1,R5
MOV R1,X        ADD C,R1        ADD C,R1        ADD C,R1        ADD B,R1
                NOP             SHL #0,R1       SHL R1,0        CMP R2,R5
                SUB #4,R1       SUB #4,R1       SUB #4,R1       SUB #4,R1
                NOP             JMP .+1         ADD R5,R5       JMP .+1
                MOV R1,X        MOV R1,X        MOV R1,X        MOV R1,X
                                                MOV R5,Y        MOV R5,Y
  (a)             (b)             (c)             (d)             (e)
```

Antivirus Techniques

- Integrity checkers
- Behavioral checkers
- Virus avoidance
  - Use good OS
  - Install **only** shrink-wrapped software
  - Use antivirus software
  - Do not click on attachments to email
  - Do frequent backups
- Recovery from virus attack
  - Halt the computer, reboot from safe disk, run antivirus software

Protection Against Viruses

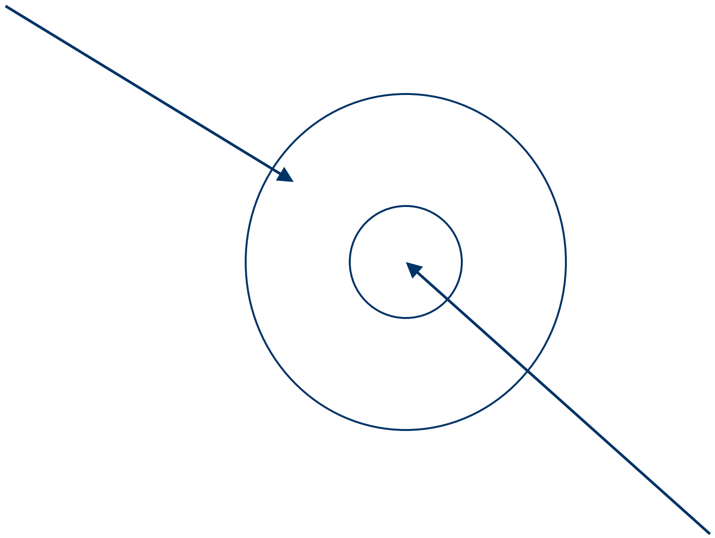
- Remember that prevention is about the *only* protection. *Buy* shrink-wrap software. Make sure you understand what you are getting into when you take the freeware.
- Remember that viruses can be distributed via Visual Basic *macro* programs in *.doc* or *.ppt* files. Be suspicious of documents in this format and insist on reformatting them in *.rtf* or print!
- Be **very** suspicious of memory sticks!
- Understand that commercial anti-virus programs look for *existing* viruses. Again, it may be too late to run one
- Start with the reformatted disk to be able to perform early detection of virus infection. Then compute a checksum for each file (and keep doing so for each new file, while recalculating the checksum for each updated file). Keep the file name/checksum encrypted
- Whenever the system is booted, compare each checksum with that in the table. Any discrepancy must rise a suspicion!

Design Principles for Security

- The system design should be public
- The default should be *no access*. (If a legitimate access is refused, it will be reported soon. Not so if an unauthorized access took place...
- Keep checking for current authority. (If you allowed me to read your file once and a week later changed the permissions, I should not be able to read it just because I kept it open for the whole week!)
- Give each process the least privilege possible
- Use only *simple* protection schemes and build them in the lowest circles of the system. Assume it is impossible to add security to an unsecure system—security is not an add-on feature!
- The scheme you chose must be acceptable to the users (alas, we are venturing into psychology), or they will not use it

Remember the *Ring Arrangement*?

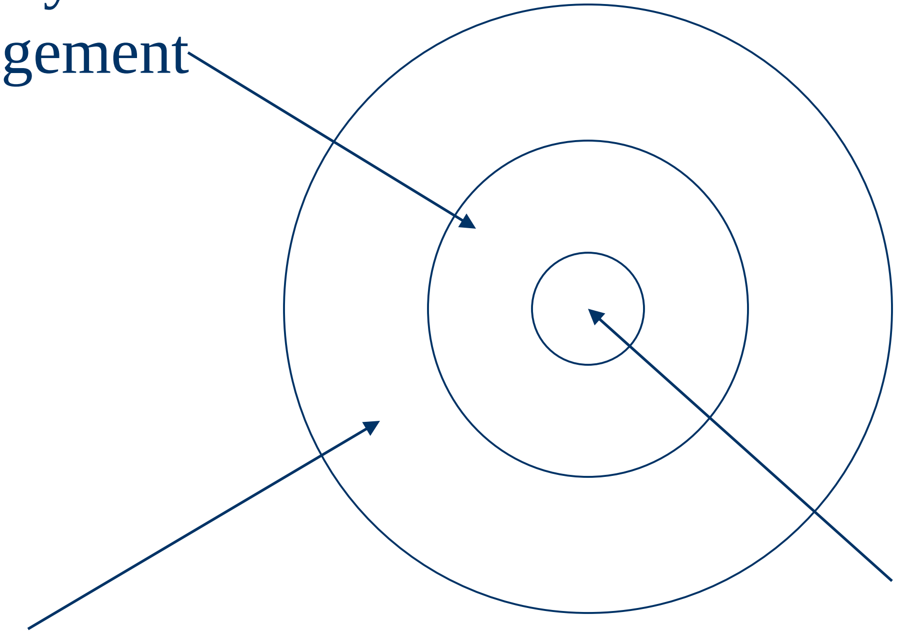
File Management



Memory Management

In this arrangement, memory management code can not use the file system

The Ring Arrangement (cont.)



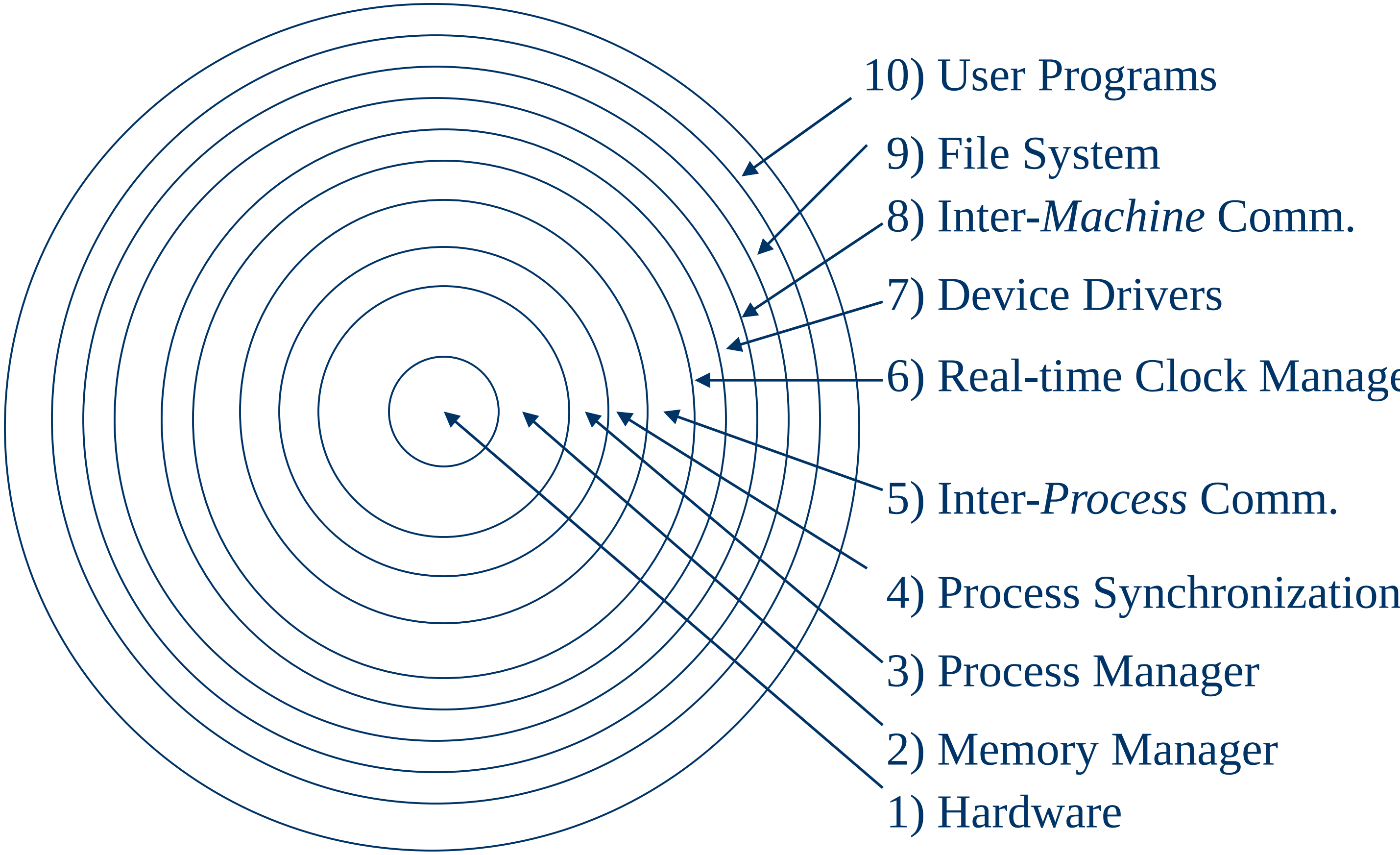
Memory Management

User Programs

Intermachine Communications

But in *this* arrangement, the memory, as seen by the user can be spread over the network!

A Good Example (XINU)



10) User Programs

9) File System

8) Inter-*Machine* Comm.

7) Device Drivers

6) Real-time Clock Manage

5) Inter-*Process* Comm.

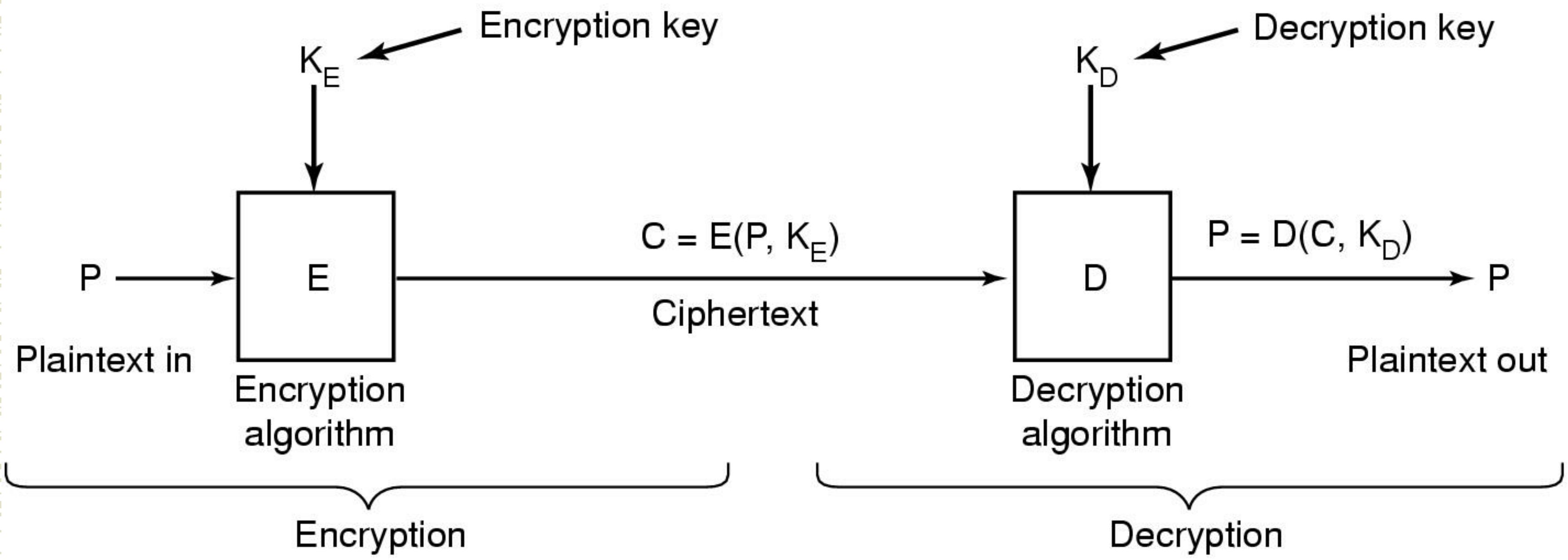
4) Process Synchronization

3) Process Manager

2) Memory Manager

1) Hardware

Basics of Cryptography



Relationship between the plaintext and the ciphertext

Secret-Key (or *Symmetric Key*) Cryptography

- Given the encryption key, K_E it is easy to find decryption key K_D
- Problem: Key distribution
  - An example: *One-Time Pad* $K_E = K_D$

⊕ (XOR)	0	1
0	0	1
1	1	0

$$x \oplus y = (\overline{x} \wedge y) \vee (\overline{y \wedge x})$$

Plaintext: 001110011010010110

Key: 100100100111110110

Cyphertext: 1 01010111101100000

Public-Key Cryptography

- All users pick a (public key, private key) pair
  - Publish the public key (= encryption key) K_E
  - Keep the private key (= decryption key) K_D secret

Two essential requirements:

1) $K_D * K_E = I$

2) It is *very hard* (i.e, *computationally infeasible)* to obtain K_D from K_E

- To send a message *M* to you, I send $K_E(M)$;
- You decrypt it, obtaining: $K_D(K_E(M)) = M.$

A Digital Signature

- …is nothing more than just an encryption of a known quantity
  - It can be used as a proof of both the origin and *integrity* of a message: (M, S[M])
  - It can be used in a *challenge/response* authentication:
    - *Challenge*: Random Quantity
    - *Response:* S(Random Quantity)
  - It can use public cryptography with any algorithm, in which $K_D * K_E = K_E * K_D$

Specificity of private-key signatures

- Consider this
  - Challenge: @3#$%^&&*##$ (= I owe you $1,000,000)
  - Response: *My_Private_Key* (@3#$%^&&*##$)
  - …

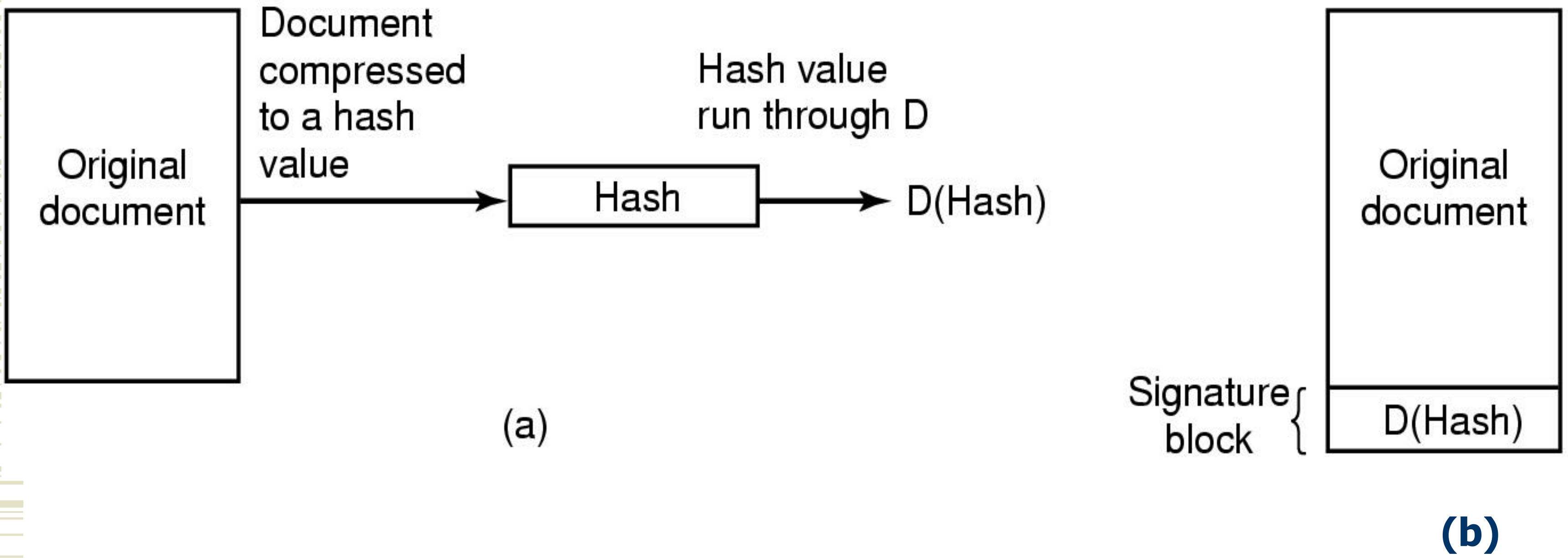
This is why signatures require a separate public key/private key pair!

One-Way Functions

- Given an algorithm for computing *f(x)*, it is *easy* to compute *y = f(x)*,
- But given the value of *y = f(x)*,

  it is *hard* (computationally infeasible) to compute *x*.
- *f(x)* typically mixes the bits of *x* thoroughly by computing a hash of x and then *signing* it:
  - *Message Digest (MD5)* produces a 16-bit result
  - *Secure Hash Algorithm (SHA)* produces a 20-byte result.

Integrity Protection with Digital Signatures



- (a) *D* is the **private key** of the sender
- (b) The receiver uses the **public key** of the sender to check the signature
- Q: How does the receiver get the public key of the sender?

Certificates

- To use this signature scheme, the sender's public key must be known
- It could be published (on a web site, for example), but then it could also be altered
- A common solution is to use *certificates:*
  - A sender attaches his or her (name, public key) pair, digitally signed by the *trusted third party*
  - Once the receiver obtained the public key of the third party, he or she can accept certificates from all senders who use this trusted third party

User Authentication

- This can be *strong* (i.e., via password) or *weak* (based on an IP address [perish the thought!], time of day [“This user will login at exactly 1:15 PM”], terminal location [If it is connected to the line that leads to the Oval Office, it must be President]
- It can also be based on *biometrics* (finger length, fingerprints, eye iris analysis, voice, etc.)

Unix Passwords

- UNIX passwords *used to be* stored in the `/etc/passwd` file as a pair: <user id>, <hashed password>
- *Linux* keeps some user information in that file, but actual hash is kept in `/etc/shadow,` which is accessible only with the root privileges
- If all passwords consisted of *seven* characters chosen at random from the 95 printable ASCII character set, the search space would be $95^7 \approx 7 \bullet 10^{13}$. At *1000* encryptions per second, it would take well over *1000* years to check them (and the whole list would fill 20 million magnetic tapes!)
- But even if a seven-letter password contains *one* lowercase character, *one* uppercase character, and *one* special character, it would still be pretty impossible to guess

Dictionary Attacks

- First, one builds a list of likely passwords (license plates, ham radio call signs, names of wives and children, birth dates, etc.) and then encrypts them (the algorithms are standard!)
- Then one compares the resulting strings with those found in the */etc/passwd* file (now unavailable!)

Salted Passwords

- Late Morris (the father) and Tompson (the father of *UNIX*) came up with a technique that would make guessing passwords pretty much useless:
  - An *n*-bit random number is associated with each password and changed every time the password is changed
  - The random number is kept in the password file in encrypted form, and the password and the random number are concatenated and then encrypted together
  - So, if I try to break into the computer account of John (who, I suspect, used his wife's name—Jill—as his password), it is not enough for me to try *jill*, *JILL*, and *Jill*. I will have to encrypt each of these in concatenation with 000, 001, 002, and so on...
  - This method is called "salting." It is used in *UNIX*. It is available now in Linux with the *Blowfish* scheme.

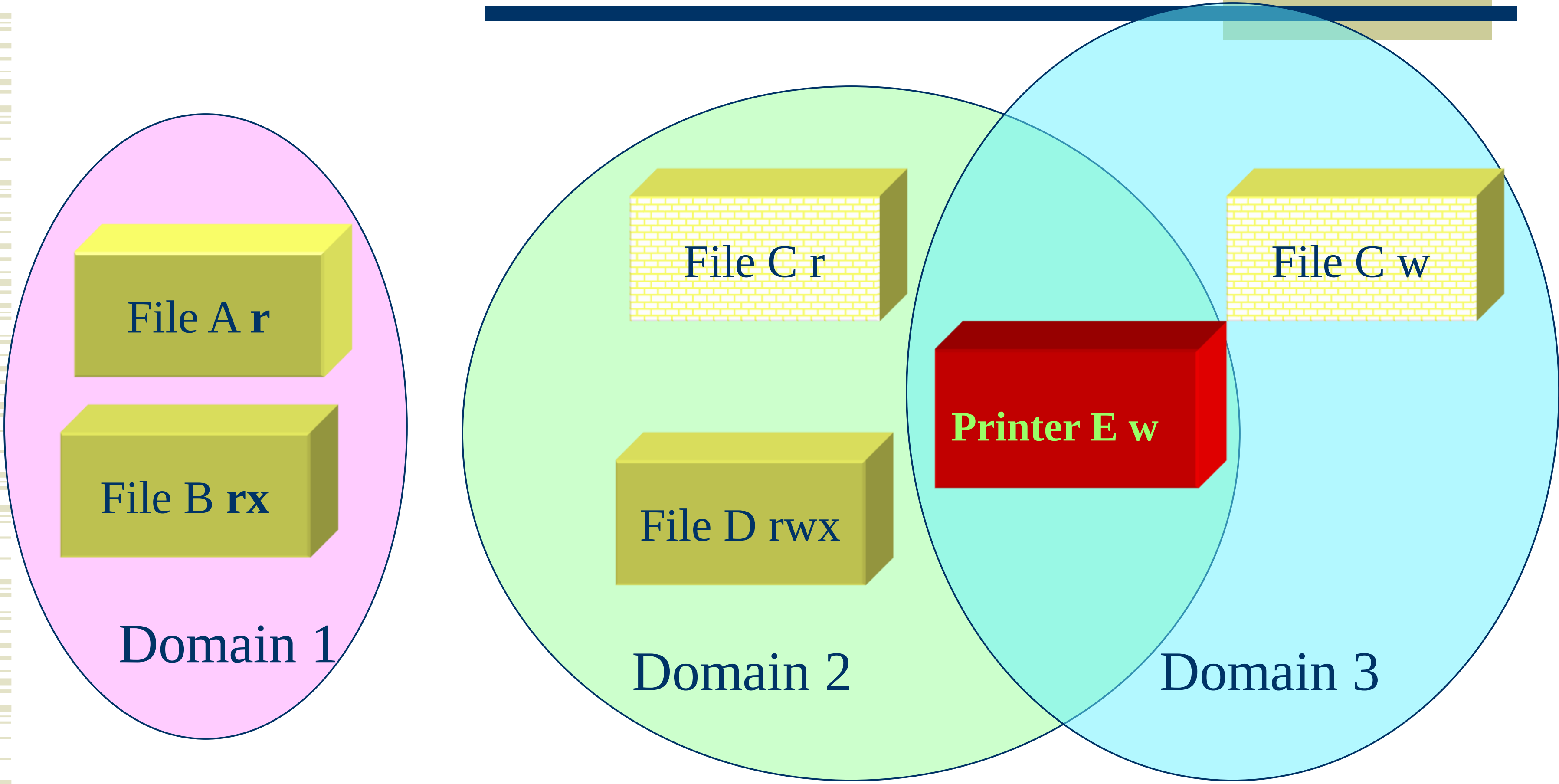
Countermeasures

- Logs (who, when)
- Baited traps (to lure the intruder and—possibly—catch the intruder)
- Keep the important machine off the network (like the *H&R Block* mainframe)—only use the terminals connected by leased telephone lines

Protection Domains

- A computer system needs to protect *objects:* hardware, memory, disk drives, semaphores, etc.
- Each such object has a *name* and *methods* (or operations)
- Each method can be executed only by the one who has the *right* to do so
- A *domain* is a set of (<object>, <rights>) pairs
    - The same object can belong (with the same or different rights) to pairs in more than one domain

An Example of Three Protection Domains



The Same Example: A Matrix Representation

	File A	File B	File C	File D	Printer E
Domain 1	r	r x			
Domain 2			r	r w x	w
Domain 3			w		w

What Is a Domain

- A process may switch domains
- In this case a domain can in itself be an object, which the process may *enter*
- A permission of a process to enter a particular domain means that the process can access all resources in the domain according to the permissions

The Same Example: The Protection Matrix Augmented

	File A	File B	File C	File D	Printer E	Domain 1	Domain 2	Domain 3
Domain 1	r	r x					Enter	
Domain 2			r	r w x	w			Enter
Domain 3			w		w			

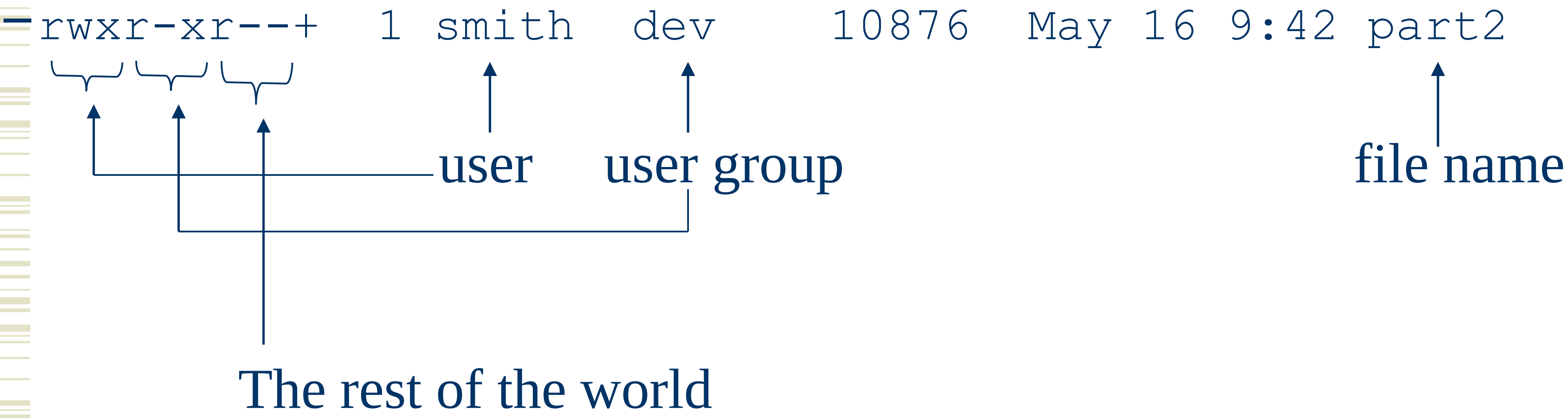
Access Control Lists

- A suitable representation for a sparse matrix (which a domain matrix would always be) is a list, in which each object would lists its domains and permissions:

File A	Domain 1	r
File B	Domain 1	r x
File C	Domain 2	r
File C	Domain 3	w
Printer E	Domain 2	w
Printer E	Domain 3	w

A *UNIX* example

- Files (and mounted devices) can belong to individual *user-ids* or *group-ids*:



```
-rwxr-xr--+   1 smith   dev     10876  May 16 9:42 part2
```

user

user group

file name

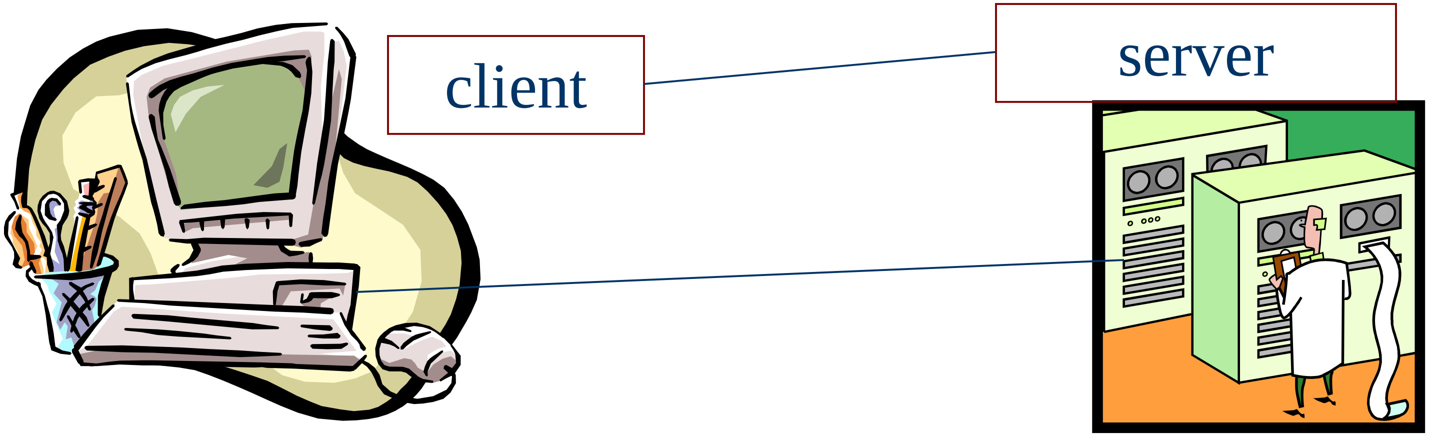
The rest of the world

Protection Models

- The *state space* (remember the bicycles and resource allocation?) of a process's access rights consists of *authorized* states and *unautorized* states
- *Protection policies* ensure that unauthorized states may not be entered
- So far it has been difficult to prove formally that a particular system (with a typical—and thus complex—set of policies) is secure

Covert Channels

- A client (on a PC) wants to talk to a server
- All communications are absolutely secure
- The server is *encapsulated* (or *confined)* so that it may not talk (or share any files or any other objects) with anyone else
- But...



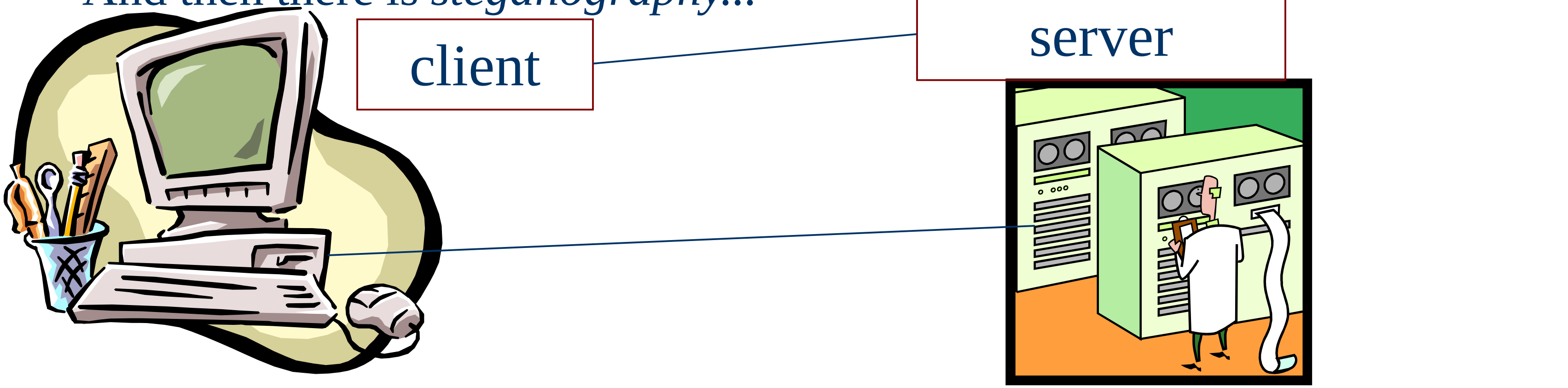
client

server

Covert Channels (cont.)

Nothing works:

- A server passes information to its collaborator
  - This can be done by using bursts of computations for *1*, and lulls for *0* (or by acquiring and releasing dedicated resources, modulating paging rate, etc....)
- This, in general, *cannot* be prevented!
- And then there is *steganography*...



Steganography *(στεγανω γραφ: covered writing)*

- The color image uses 1024 * 769 *picture cells (pixels)*
- Each pixel consists of three 8-bit numbers (RGB): {red intensity, green intensity, blue intensity}
- Stealing one bit from each color (7-bit color is practically undistinguishable from 8-bit color), one gets 1024*769*3/8 = 294,912 bytes to store secret information (which can also be compressed and encrypted)

Steganography (example)

Me

Me… and my Lecture 1

Using *S-tools (Steganography tools for Windows by A. Brown. Freeware,* available at http://www.spychecker.com/program/stools.html*)*

Summary

- An OS can be threatened by insider and outsider attacks
- Authentication (passwords, in particular, but also smart cards and biometric devices) is an important protection device
- Viruses, worms, and mobile code attacks have increasing
- Secure systems can be designed
- The use of *secure boot* rooted in the hardware must be mandatory
- But even provably secure systems cannot prevent the use of covert channels