Chapter 9

Security

9.1 The security environment
9.2 Basics of cryptography
9.3 User authentication
9.4 Attacks from inside the system
9.5 Attacks from outside the system
9.6 Protection mechanisms
9.7 Trusted systems

SECURITY

Based on the slides of Tanenbaum and modified by Albert Levi

The Security Environment

Threats

Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service

Security goals and threats

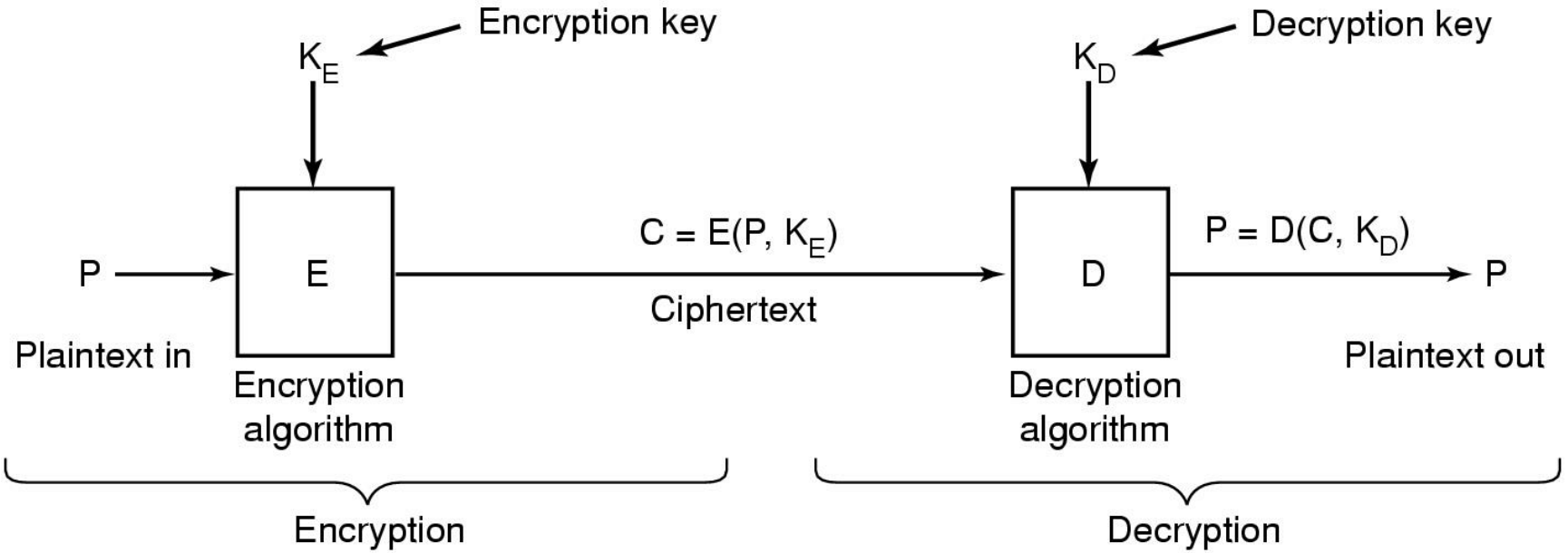
Intruders

- Active vs. Passive Intruders
- Common Categories
  1. Casual prying by nontechnical users
  2. Snooping by insiders
  3. Determined attempt to make money
  4. Commercial or military espionage
- Precautions must be economically justified

Hackers vs. Crackers

- Hacker
  - a great programmer, who knows too much about computers
    - may use this ability for bad, but this is not often
- Cracker
  - bad guy who tries to break into computer systems

Basics of Cryptography



Relationship between the plaintext and the ciphertext

Secret-Key Cryptography

- Secret-key crypto called symmetric-key crypto
  - the same key is used for both encryption and decryption
- An example: monoalphabetic substitution
  - each letter replaced by different letter
  - is it safe? Let's analyze and discuss

- Some Modern Symmetric Cryptosystems
  - DES (a bit old), 3DES (being discarded, but still in use), AES (new standard)

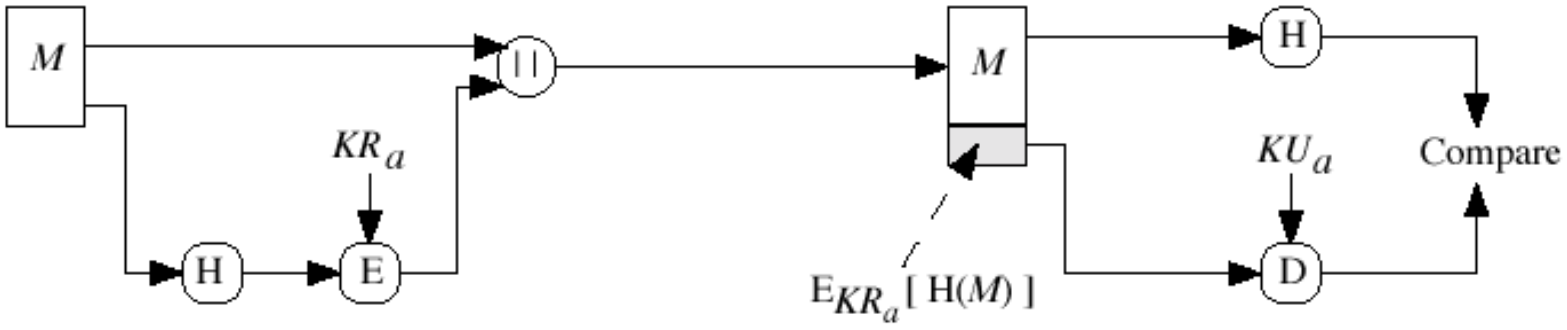
Public-Key Cryptography

- All users pick a public key/private key pair
  - publish the public key
  - private key not published
  - they are mathematically related but it is not feasible to obtain private key given public one
- Public key is the encryption key
  - anyone can encrypt
- private key is the decryption key
  - Only the owner can decrypt
- Based on mathematically intractable problems like discrete log and factorization of large numbers
  - RSA, Diffie-Hellman are examples

One-Way Functions

- Function such that given formula for f(x)
  - easy to evaluate y = f(x)
- But given y
  - computationally infeasible to find x
- Standard Hash functions
  - MD5 (not so popular nowadays)
    - 128-bit output
  - SHA1 (NIST standard and commonly used)
    - 160-bit output

Digital Signatures



M: message to be signed H: Hash function

E: RSA Private Key Operation KR_a: Sender's Private Key

D: RSA Public Key Operation KU_a: Sender's Public Key

$E_{KRa}[H(M)]$ Signature of A over M

- Problem: How can we get authentic KUa?
  - certificates

User Authentication

Basic Principles. Authentication must identify:

1. Something the user knows
2. Something the user has
3. Something the user is

This is done before user can use the system

Authentication Using Passwords

LOGIN: ken
PASSWORD: FooBar
SUCCESSFUL LOGIN

(a)

LOGIN: carol
INVALID LOGIN NAME
LOGIN:

(b)

LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:

(c)

(a) A successful login

(b) Login rejected after name entered

(c) Login rejected after name and password typed

Password Guessing

- Exhaustive Search (Brute Force)
  - try all possible combinations
  - may work if the symbol space and password length are small
- Intelligent Search
  - search possible passwords in a restricted space
    - related to the user: girlfriend/boyfriend name, car brand, phone number, birth date, …
    - generic: meaningful words or phrases, dictionary attack
- War Dialers and ping attacks can be used to find victims
  - telnet may be used for on-line attacks.

How a cracker broke into LBL*

LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL

*Lawrence Berkeley Lab, a U.S. Dept. of Energy research lab

How to choose a password

- "Have" a password
  - do not let it blank
- Do not use default passwords, change them ASAP
  - like "pass"
- Use mixed symbols
  - upper and lowercase letters, digits, symbols
- use long passwords
- avoid meaningful and obvious words and their derivatives
  - e.g. RoseGarden1, Saygin123
- A useful mechanism: Pick a phrase or sentence and use initials as password
  - e.g. "I hate when system asks me to change password" → Ihwsam2cp

How the system helps?

- Sysadmin can try to guess a password with known techniques
- Password ageing
  - users are enforced to change their passwords periodically
  - possibly by prohibiting to use old passwords
- Limit login attempts
  - temporarily blocks the account
- Inform user
  - about last successful login time and number of unsuccessful attempts
  - automatic callback at number prespecified
- Login logs

Average user behavior

- They do not memorize long and random password
  - instead they prefer to write down passwords
- they tend to derive passwords from the old one
  - e.g. by adding 1, 2, …
  - guessing one makes easier to guess others
- They prefer not to change or revert back to their original password
  - so it is not a good idea to enforce them to change passwords too often

Rule of thumb

"Enforcing too much security may weaken the system, since the users tend to circumvent security rules to do their job properly"

Password Spoofing

- Are you really talking to the server that you want to talk
  - fake login prompts
    - when you try to login a shared station
      - previous user may leave a fake login screen
    - How to avoid detect
      - unsuccessful login reports
      - Ctrl-Alt-Del in Windows NT and 2000
- remote login is even worse,
  - telnet sends passwords in clear
  - use SSH (Secure Shell)

Password Storage

- Passwords should be able to be verified by the server
  - so the passwords should be stored, but how?
- Passwords are generally stored in encrypted form
  - using symmetric encryption or one-way hash functions
- Possible *off-line* attack
  - Even if the passwords are stored in encrypted form, dictionary attacks are possible when the file contains the encrypted passwords is obtained by the attacker
  - this is a passive off-line attack
    - unsuccessful attempt limits do not help

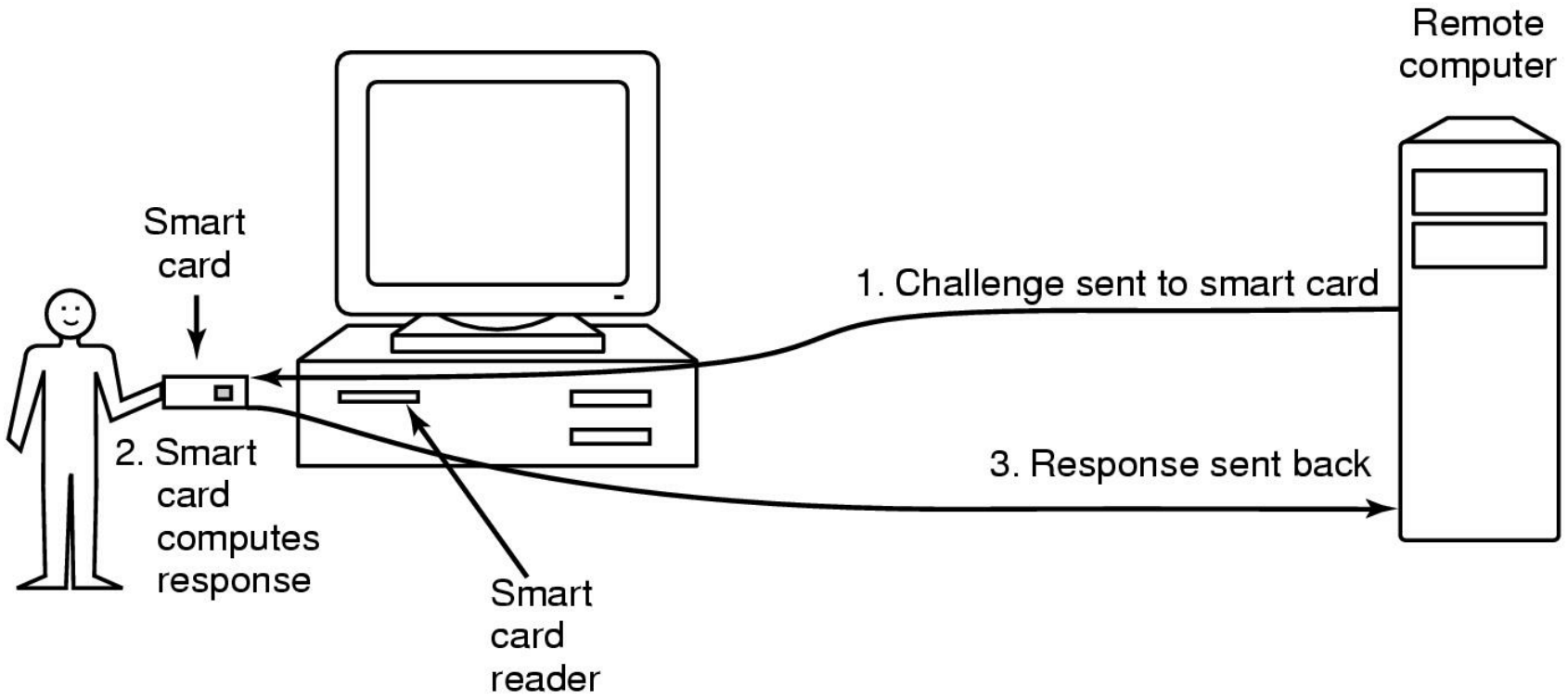
How to prevent dictionary attacks on password files – 1

- Slow down password encryption
  - UNIX crypt function
    - repeats a modified version of DES 25 times
    - on all-zero block data and using the password as the key
- Do not make the password file publicly readable
  - shadow passwd file in UNIX systems

How to prevent dictionary attacks on password files - 2

- Password Salting
  - Encrypt passwords with additional random value (salt)
  - salt is not a secret value
  - store the encrypted password with salt
  - Salting slows down dictionary attack
    - since the attacker should run a brand new dictionary search for each user
  - Another advantage
    - if two users have the same password, their encrypted passwords will not be same (of course if the salt values are not accidentally the same)

Authentication Using a Physical Object



- e.g. Plastic cards
  - magnetic stripe cards
  - chip cards: stored value cards, smart cards
- can be stolen or lost
  - should be used together with a PIN or password

Biometric Authentication

- Uses unique biological properties like
  - fingerprint
  - palm print
  - retina pattern
- does not have 100% accuracy
  - false accept
    - should reject, but accepts - very bad
  - false reject
    - should accept, but rejects
    - not so bad but inefficient systems are not used
  - trade-off between false accept and false reject
- two controversies
  - if copied or broken, you cannot change it
  - people may not like their fingerprints are taken as criminals or laser beams in their eyes

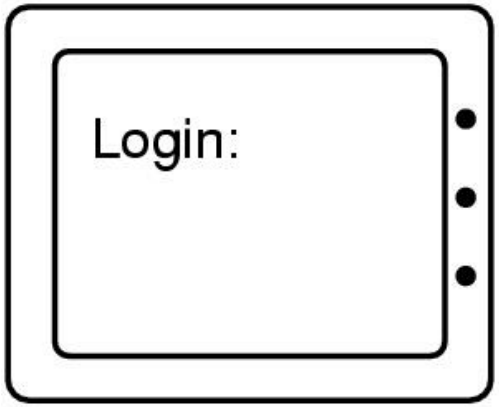
Other Authentication Approaches

- What you do
  - mechanical tasks that have specific properties that only you can do
- Dynamic signatures
  - pressure, speed, orientation are properties as well as the shape
- Keyboard typing
  - speed, intervals between keystrokes
  - false accept, false reject problems exist here too

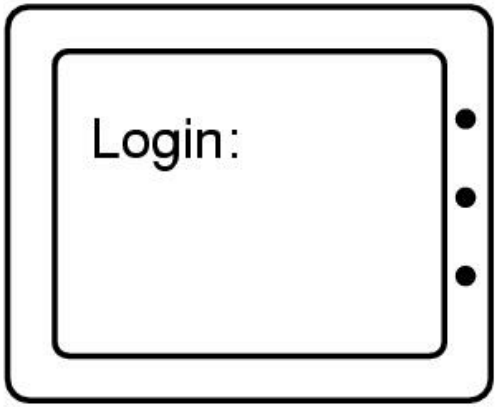
Operating System Security
Trojan Horses

- Free program made available to unsuspecting user
  - Actually contains code to do harm
- Place altered version of utility program on victim's computer
  - trick user into running that program

Login Spoofing



(a)



(b)

(a) Correct login screen

(b) Phony login screen

Trap Doors

```
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)

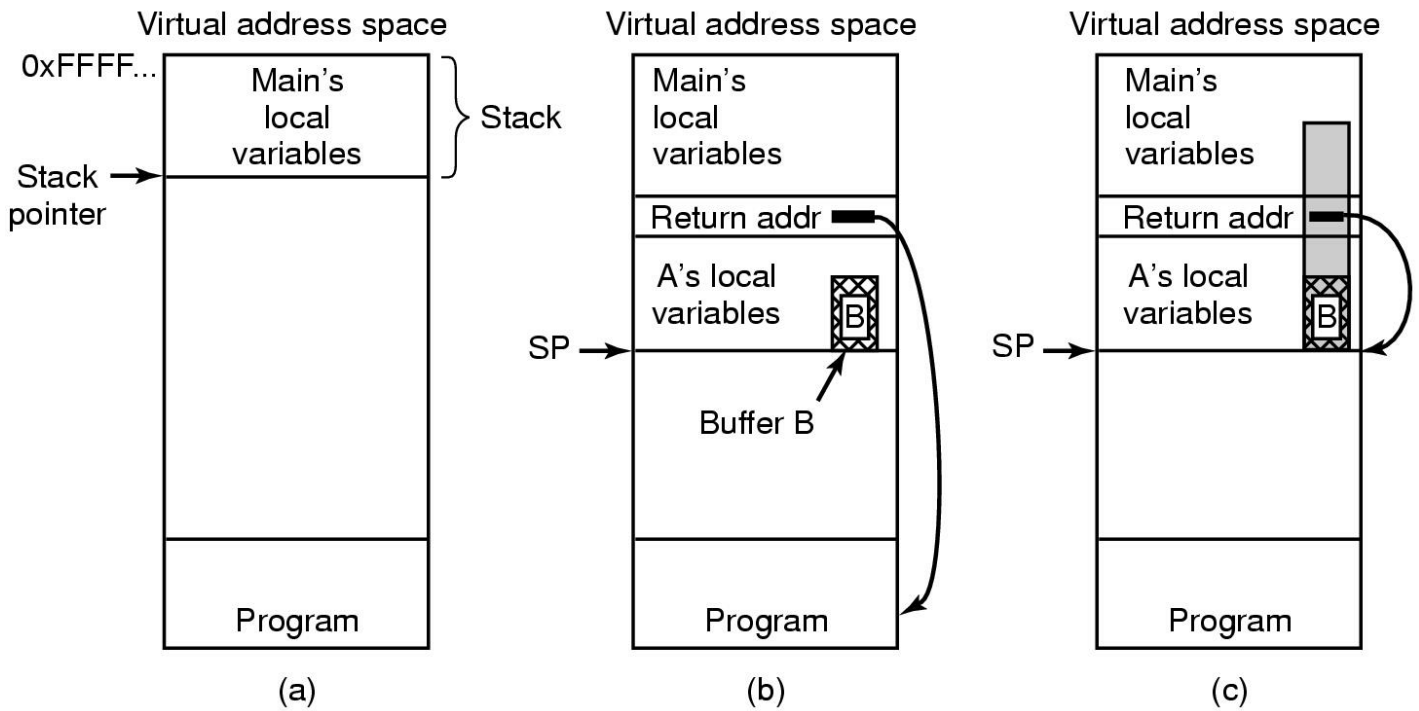
```
while (TRUE) {
        printf("login: ");
        get_string(name);
        disable_echoing( );
        printf("password: ");
        get_string(password);
        enable_echoing( );
        v = check_validity(name, password);
        if (v || strcmp(name, "zzzzz") == 0) break;
}
execute_shell(name);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

(b)

(a) Normal code. (b) Code with a trapdoor inserted

SOLUTION: Software companies should enforce peer-to-peer or group code reviews

Buffer Overflow



(a) Situation when main program is running

(b) After program *A* called

(c) Buffer overflow shown in gray

Buffer overflow is a well-known problem for fixed-size strings and arrays. The input into those variables may overflow and overflowing part may be a malicious program. Do not use *gets* and try to use dynamic allocation in C programs

Design Principles for Security

1. System design should be public
2. Default should be "no access"
3. Give each process least privilege possible
4. Protection mechanism should be
   - simple
   - uniform
   - in lowest layers of system
5. Scheme should be psychologically acceptable

And … simplest is the best

Attacks from Outside - Viruses

- Virus = program can reproduce itself
  - attach its code to another program
  - additionally, does harm
- Goals of virus writer
  - quick spreading
  - difficult to detect
  - hard to get rid of

Virus Damage Scenarios

- Harmless stuff
- Blackmail
- Denial of service as long as virus runs
- Permanently damage hardware (BIOS on flash)
- Target a competitor's computer
  - do harm
  - espionage
- Intra-corporate dirty tricks
  - sabotage another corporate officer's files

How Viruses Work (1)

- Mostly written in assembly language
- Inserted into another program
  - use tool called a “dropper”
- Virus dormant until program executed
  - then infects other programs
    - recursively searches the file system and infects all possible files
  - eventually executes its “payload”

How Viruses Work (2)

Recursive procedure that finds executable files on a UNIX system

Virus could infect them all, but this is not a good practice. Why?

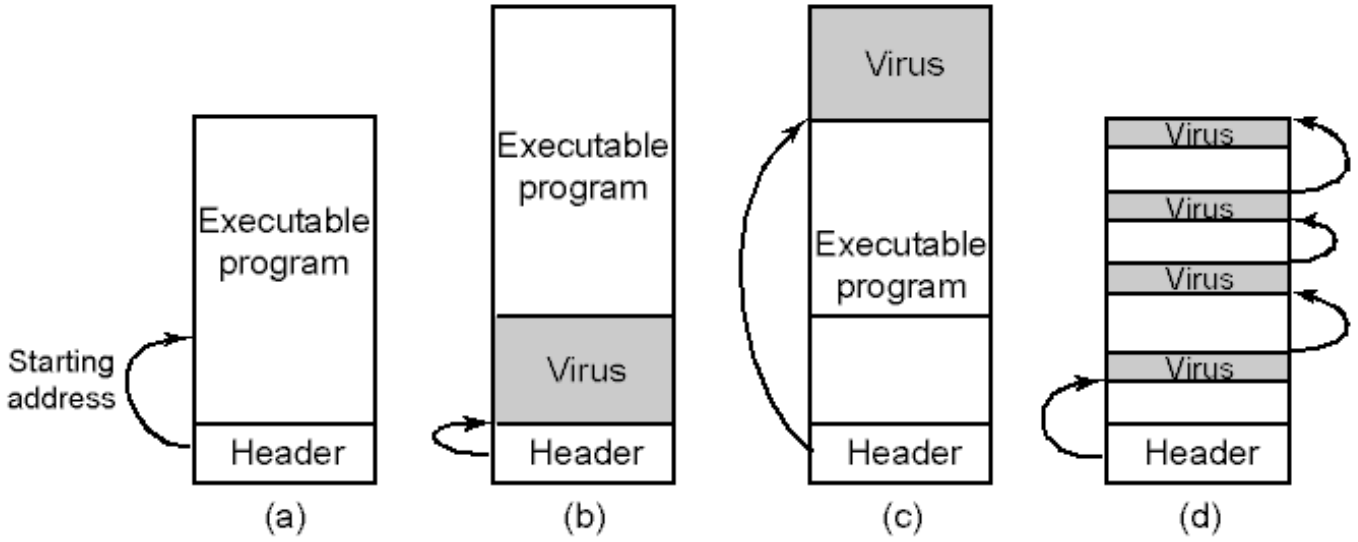
Infected files should not be infected again. Why?

```
#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 Work (3)

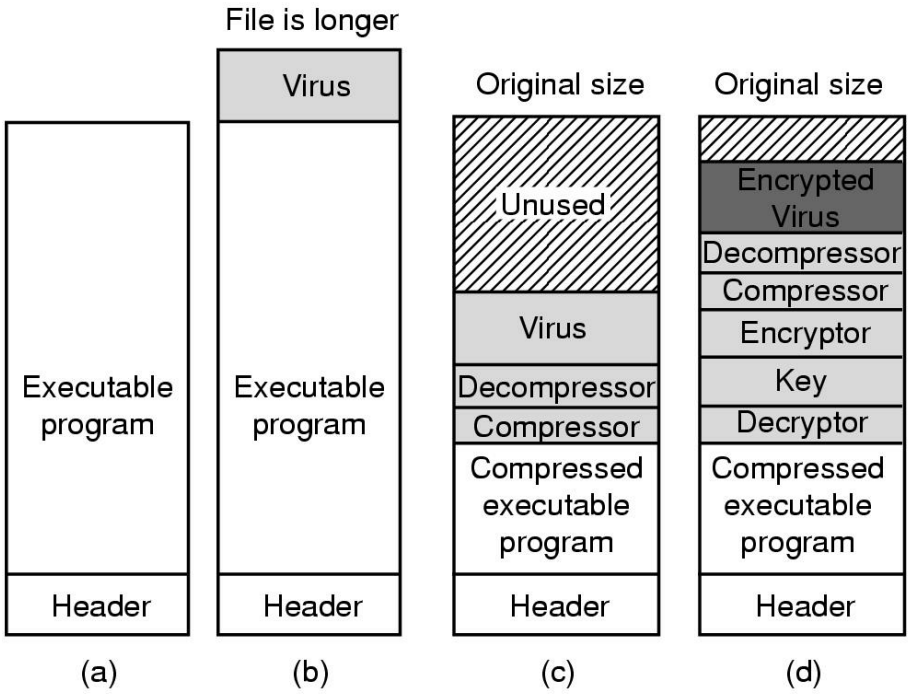


- An executable program
- With a virus at the front
- With the virus at the end
- With a virus spread over free space within program (cavity virus) – does not change the size of the program

Antivirus and Anti-Antivirus Techniques

- cat-and-mouse game
  - viruses try to hide themselves
  - antivirus software tries to catch
- Virus Scanners
  - virus database that contains the virus codes and characteristics
  - files are checked against this database
    - a fuzzy search is needed to catch variants
  - scan-only-changed-files is a good performance improving technique
    - but how can you understand the modified files?
    - modification date check – does it work? Not really!
    - size control? See next slide

Antivirus and Anti-Antivirus Techniques



(a) A program
(b) Infected program
(c) Compressed infected program
(d) Encrypted virus with random key for each infected file

still cannot be hidden

Antivirus and Anti-Antivirus Techniques

```
MOV A,R1
ADD B,R1
ADD C,R1
SUB #4,R1
MOV R1,X
```
(a)

```
MOV A,R1
NOP
ADD B,R1
NOP
ADD C,R1
NOP
SUB #4,R1
NOP
MOV R1,X
```
(b)

```
MOV A,R1
ADD #0,R1
ADD B,R1
OR R1,R1
ADD C,R1
SHL #0,R1
SUB #4,R1
JMP .+1
MOV R1,X
```
(c)

```
MOV A,R1
OR R1,R1
ADD B,R1
MOV R1,R5
ADD C,R1
SHL R1,0
SUB #4,R1
ADD R5,R5
MOV R1,X
MOV R5,Y
```
(d)

```
MOV A,R1
TST R1
ADD C,R1
MOV R1,R5
ADD B,R1
CMP R2,R5
SUB #4,R1
JMP .+1
MOV R1,X
MOV R5,Y
```
(e)

- Examples of a polymorphic virus
- All of these examples do the same thing
- Generated by “mutation engines” automatically
- Hard to detect, but there are not so many such engines

Antivirus and Anti-Antivirus Techniques

- Integrity checkers
- Behavioral checkers – memory resident
- Virus avoidance
  - use antivirus software
  - do not click on attachments to email
  - frequent backups
- Recovery from virus attack
  - halt computer, reboot from safe disk, run antivirus

Sandboxing

- For running untrusted code such as Java Applets
- Confine the program into a limited address-space and operations
- Use a reference monitor to check actions

Interpretors

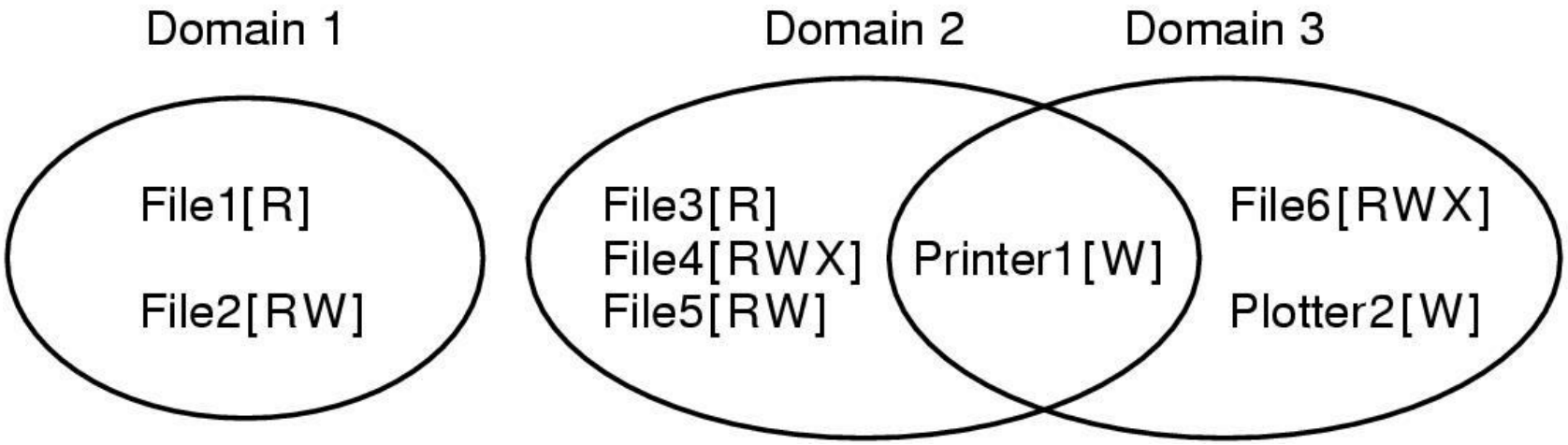
- Ex: Java interpretor
- For untrusted code: check each system call and memory reference
- For trusted code: proceed without checking
- How to make sure the code is trusted?

Protection Mechanisms

- Policies
  - whose data to be protected from whom
- Protection Mechanism
  - how the system enforces these policies
- Reference Monitor
  - a program that checks legality of the access requests
  - has several components

Protection Mechanisms

Protection Domains (1)



- A domain is a set of (object, rights) pairs
- Objects can be software (e.g. files, processes) or hardware (e.g. printers)
- Rights: permissions to operate on object (e.g. read, write, execute)

Protection Domains in UNIX

- The domain of a process is defined by userID (UID) and GroupID (GID)
  - each (UID,GID) pair corresponds to a list of objects and access rights

Protection Domains (2)

Implementation issues

	Object							
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2
Domain 1	Read	Read Write						
2			Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

A protection matrix

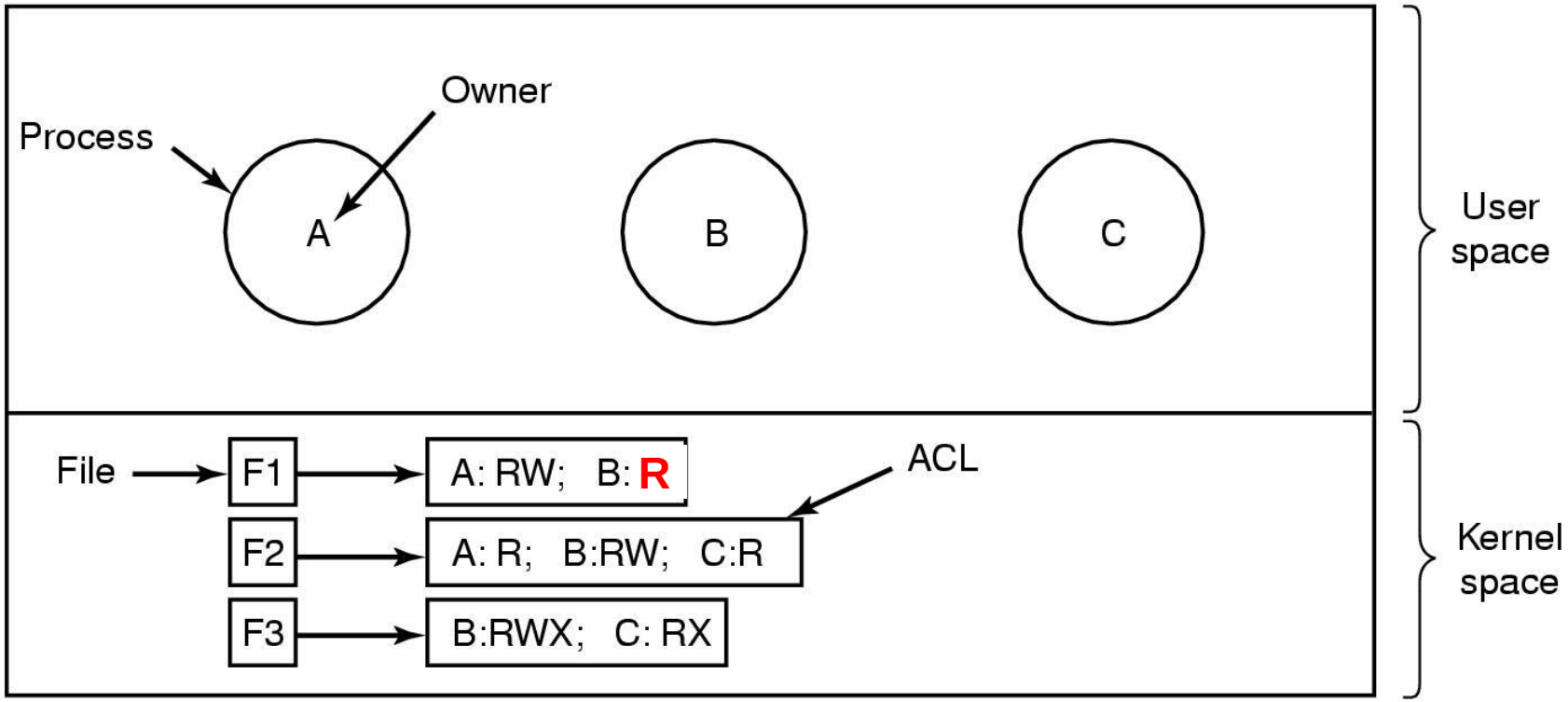
(actually not a common method)

Protection Domains (3)

	Object									
	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
main 1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

- A protection matrix with domains as objects
- Useful for domain switching
  - e.g. kernel part of UNIX processes (system calls)
    - kernel runs in another domain

Access Control Lists (1)



- Use of access control lists of manage file access
- Simplistic assumptions of this example: each domain belongs to a single user, simple access rights (rwx)

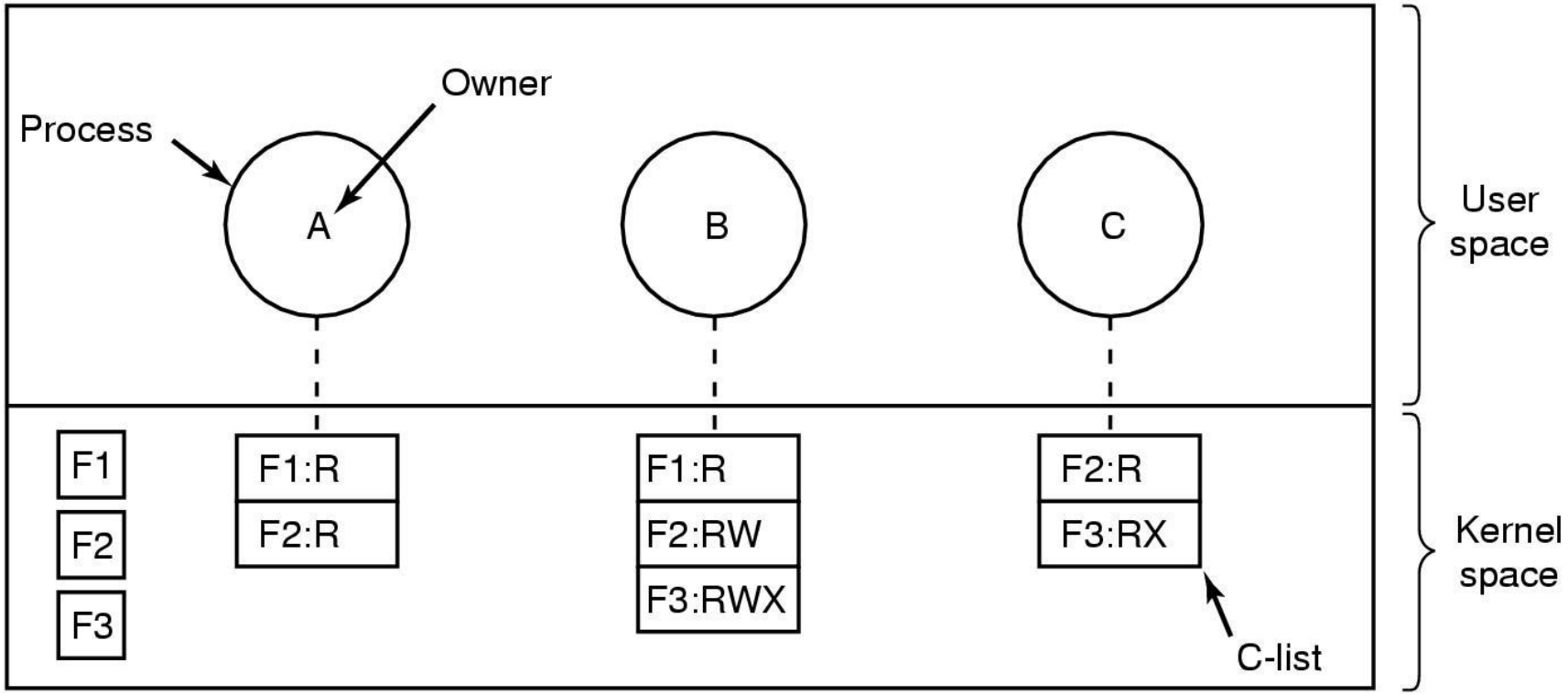
Access Control Lists (2)

- Complicated access rights such as append, delete, copy, etc.
- Groups are also incorporated into ACLs

  UID1, GID1: rights1; UID2, GID2: rights2; …
- Groups can be modeled as *roles* and access rights are given to roles

File	Access control list
Password	tana, sysadm: RW
Pigeon_data	bill, pigfan: RW; tana, pigfan: RW; ...

Capabilities (1)



Each process has a capability list (C-list)

Capabilities (2)

- C-lists are also objects, so they have to be protected from malicious tampering by the user
- Cryptographically-protected capability

Server	Object	Rights	f(Objects, Rights, Check)

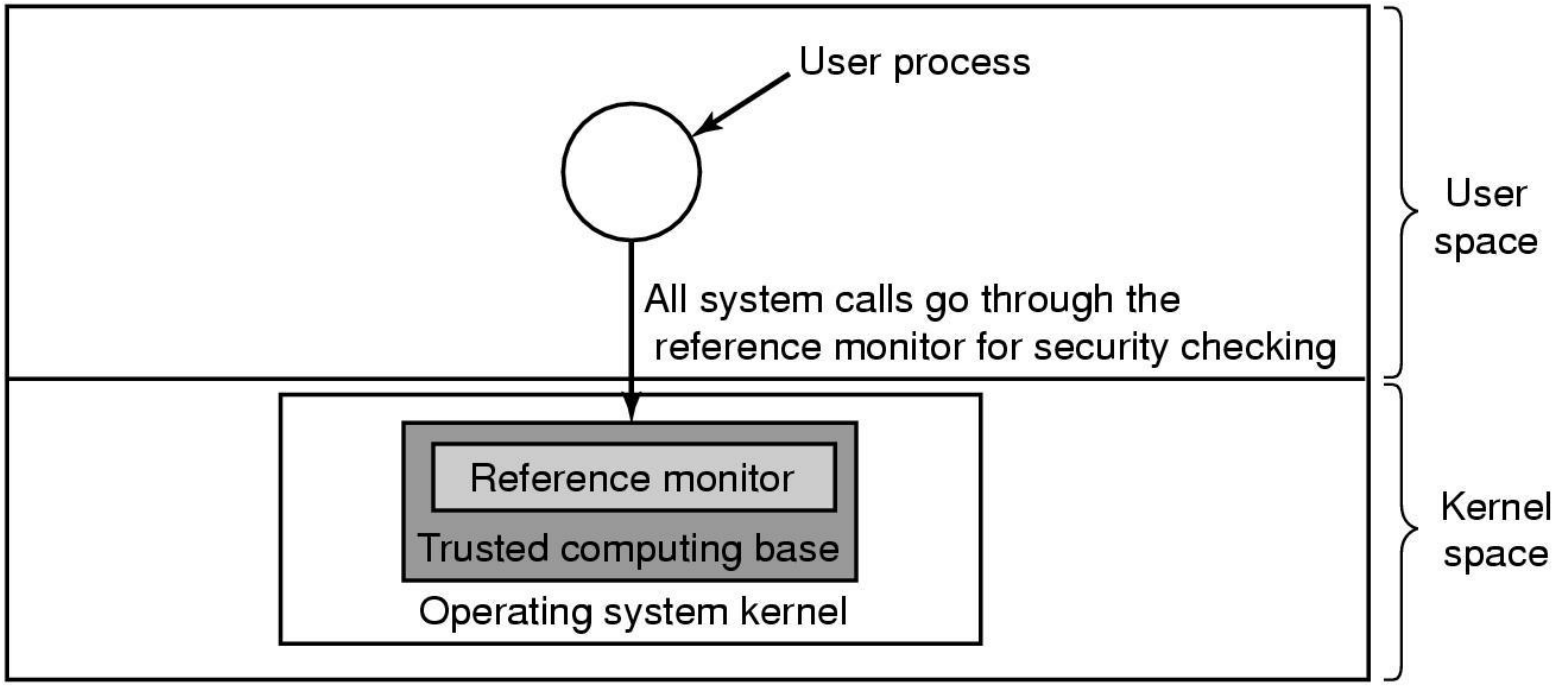
- Rights cannot be modified by the user
- Revocation of capabilities is a problem
  - indirect objects
  - changing the check field at server or file system

Are we going to have a secure OS?

- It is very easy to write an OS that is immune to viruses
  - just disallow any executables to run
  - Would you use it?
- Typical user nature (and also an economic fact)
  - MORE FEATURES
    - => more complexity => more code => more bugs and holes => more security breaches

Trusted Systems

Trusted Computing Base (TCB) and reference monitor



– Trusted System: a system in which specific security requirements are defined and met
– TCB: in which all security rules are strictly enforced with no exceptions. In UNIX, root programs, process and memory management are in TCB.
– Reference Monitor is in TCB

Formal Models of Secure Systems

	Objects		
	Compiler	Mailbox 7	Secret
Eric	Read Execute		
Henry	Read Execute	Read Write	
Robert	Read Execute		Read Write

(a)

	Objects		
	Compiler	Mailbox 7	Secret
Eric	Read Execute		
Henry	Read Execute	Read Write	
Robert	Read Execute	Read	Read Write

(b)

(a) An authorized state (b) An unauthorized state

- Suppose there is set of commands to change the access rights and Robert has managed to run a process to end up with (b)

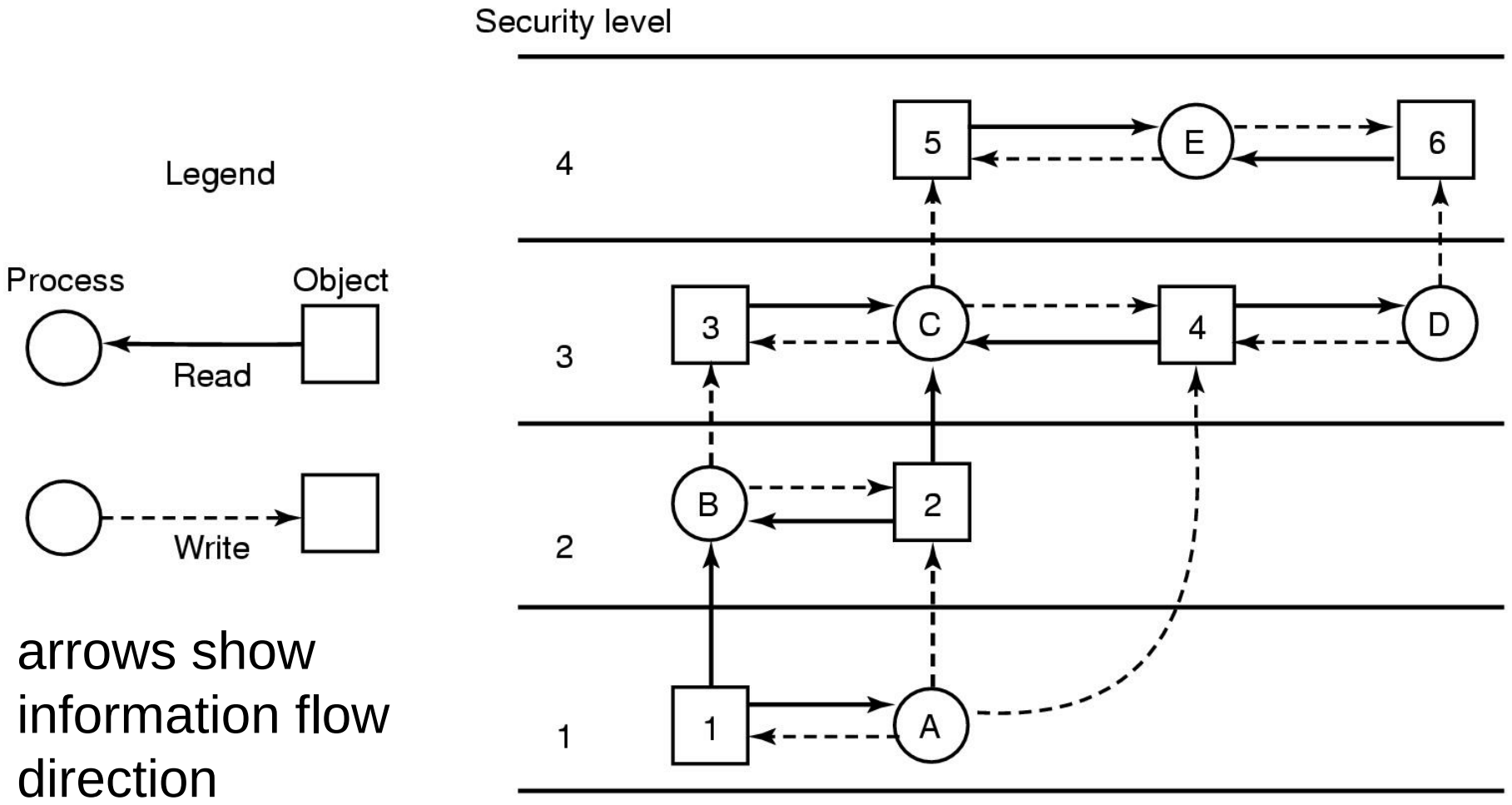
Access Control Policies

- Discretionary Access Control
  - individual users determine the fate of their objects
- Mandatory Access Control
  - tighter than discretionary
  - system level controls that may not be altered or by-passed by individual users
  - AIM: regulate the information flow in secure way

Multilevel Security (1)

- Bell-La Padula Model (1973)
  - originally for military
  - objects (documents) have different security levels
    - unclassified, confidential, secret, top secret
  - likewise the people (depending on what type of docs that they can see)
- Rules of BLP Model
  - Simple security property (no-read-up): a process can read objects only at its level or below
  - * property (no-write-down): A process can write objects only at its level or higher
  - reverses are not possible. So, information cannot leak from a high security level to a lower one.

Multilevel Security (2)



arrows show information flow direction

The Bell-La Padula multilevel security model

Multilevel Security (3)

The Biba Model

- BLP is good for military to keep the secrets, but what about the integrity?
  - accountant should not write president's files
- Principles to guarantee integrity of data
  1. Simple integrity principle (no-write-up)
     - process can write only objects at its security level or lower
  2. The integrity * property (no-read-down)
     - process can read only objects at its security level or higher

Orange Book Security (1)

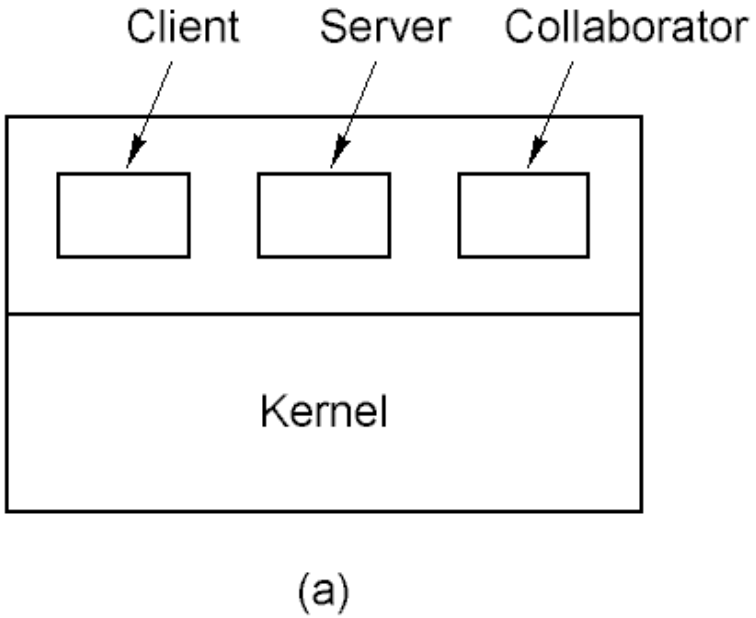
Criterion	D	C1	C2	B1	B2	B3	A1
Security policy							
Discretionary access control		X	X	→	→	X	→
Object reuse			X	→	→	→	→
Labels				X	X	→	→
Label integrity				X	→	→	→
Exportation of labeled information				X	→	→	→
Labeling human readable output				X	→	→	→
Mandatory access control				X	X	→	→
Subject sensitivity labels					X	→	→
Device labels					X	→	→
Accountability							
Identification and authentication		X	X	X	→	→	→
Audit			X	X	X	X	→
Trusted path					X	X	→

- Symbol X means new requirements
- Symbol -> requirements from next lower category apply here also

Orange Book Security (2)

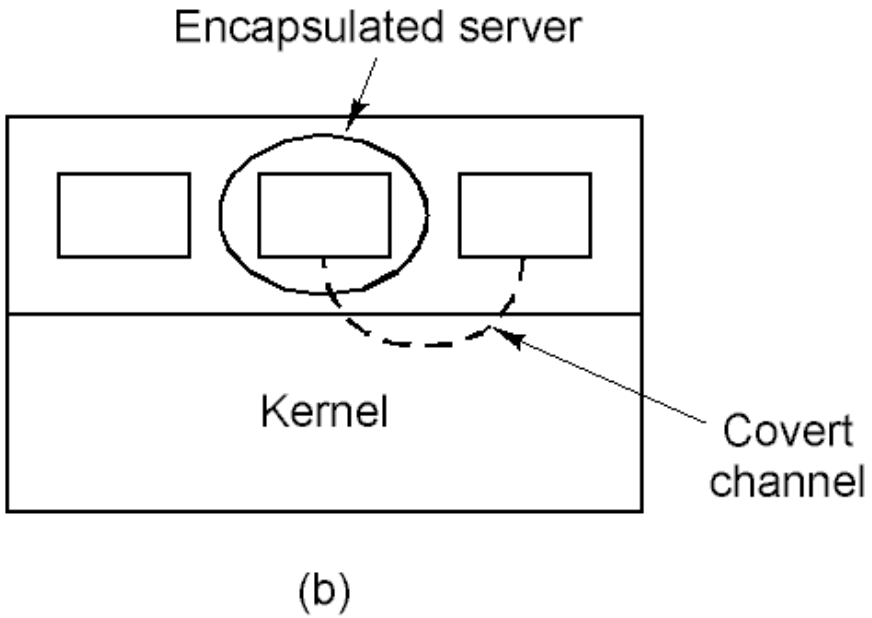
Assurance							
System architecture		X	X	X	X	X	→
System integrity		X	→	→	→	→	→
Security testing		X	X	X	X	X	X
Design specification and verification				X	X	X	X
Covert channel analysis					X	X	X
Trusted facility management					X	X	→
Configuration management					X	→	X
Trusted recovery						X	→
Trusted distribution							X
Documentation							
Security features user's guide		X	→	→	→	→	→
Trusted facility manual		X	X	X	X	X	→
Test documentation		X	→	→	X	→	X
Design documentation		X	→	X	X	X	X

Covert Channels (1)



(a)

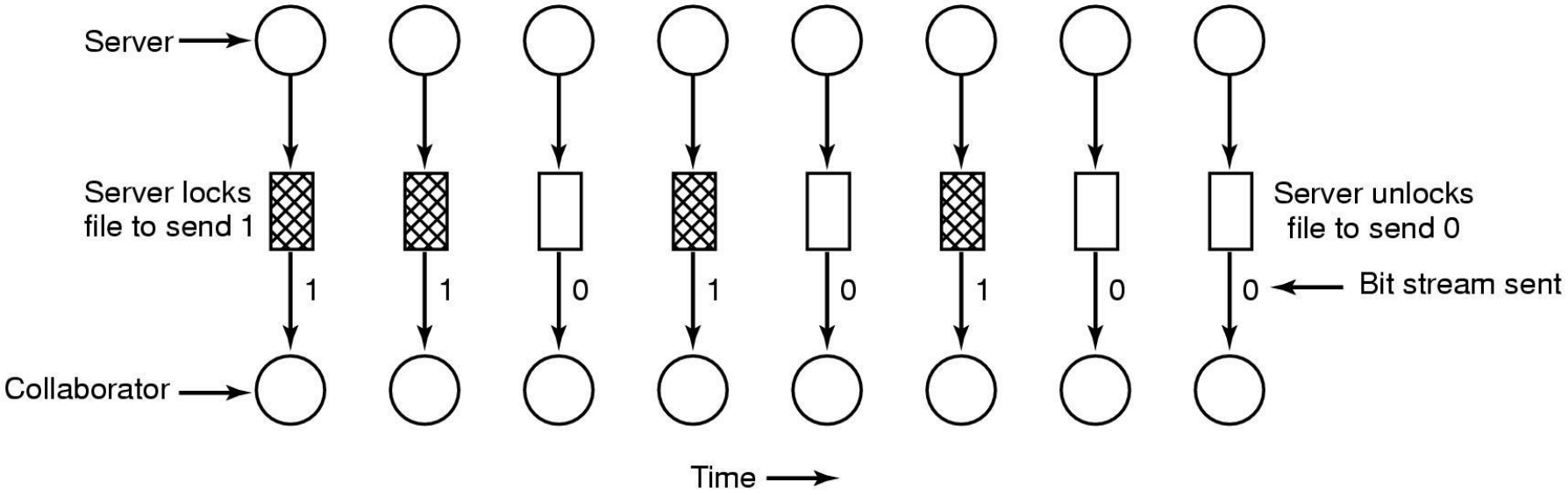
Client, server and collaborator processes



(b)

Encapsulated server can still leak to collaborator via covert channels

Covert Channels (2)



A covert channel using file locking

Covert Channels (3)

- Pictures appear the same
- Picture on right has text of 5 Shakespeare plays
  - encrypted, inserted into low order bits of color values

Zebras

Hamlet, Macbeth, Julius Caesar
Merchant of Venice, King Lear