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

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

Common Categories

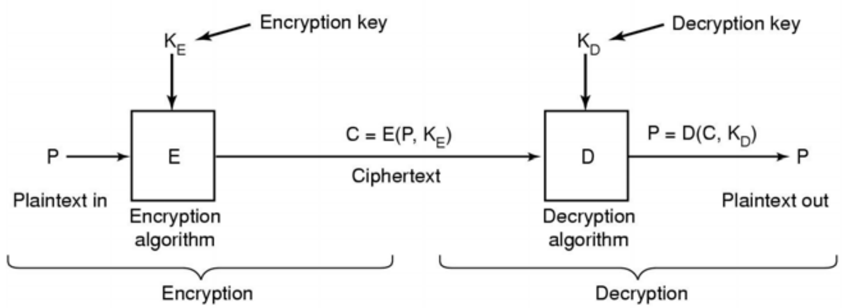
1. Casual prying by nontechnical users
2. Snooping by insiders
3. Determined attempt to make money
4. Commercial or military espionage

Accidental Data Loss

Common Causes

1. Acts of God
   - fires, floods, wars
2. Hardware or software errors
   - CPU malfunction, bad disk, program bugs
3. Human errors
   - data entry, wrong tape mounted

Basics of Cryptography



Relationship between the plaintext and the ciphertext

Secret-Key Cryptography

- Monoalphabetic substitution
  - each letter replaced by different letter
- Given the encryption key,
  - easy to find decryption key
- Secret-key crypto called symmetric-key crypto

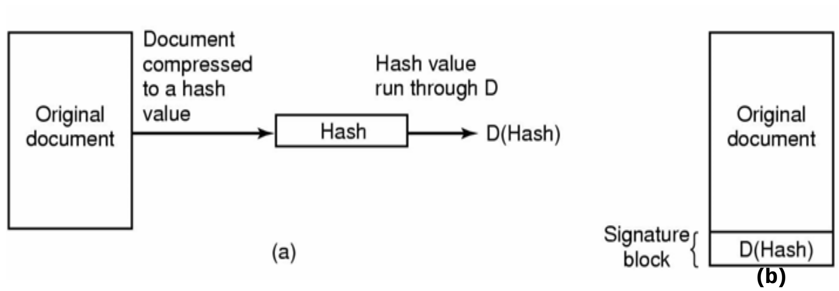
Public-Key Cryptography

- All users pick a public key/private key pair
  - publish the public key
  - private key not published
- Public key is the encryption key
  - private key is the decryption key

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

Digital Signatures



- Computing a signature block
- What the receiver gets

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

Authentication Using Passwords

```
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
```

- How a cracker broke into LBL
  - a U.S. Dept. of Energy research lab

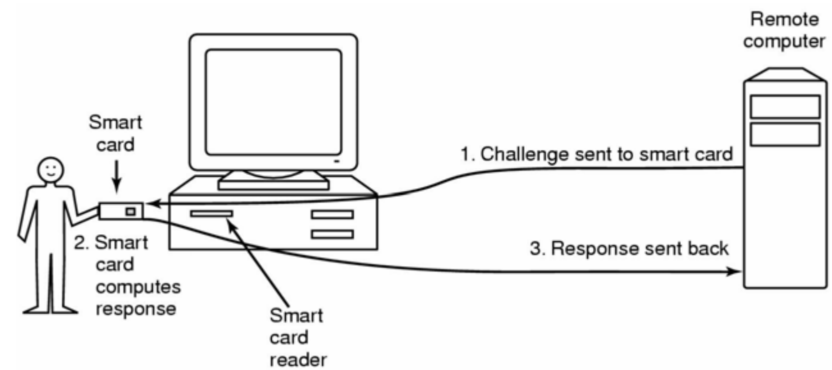
Authentication Using Passwords

Bobbie, 4238, e(Dog4238)
Tony, 2918, e(6%%TaeFF2918)
Laura, 6902, e(Shakespeare6902)
Mark, 1694, e(XaB@Bwcz1694)
Deborah, 1092, e(LordByron,1092)

Salt Password

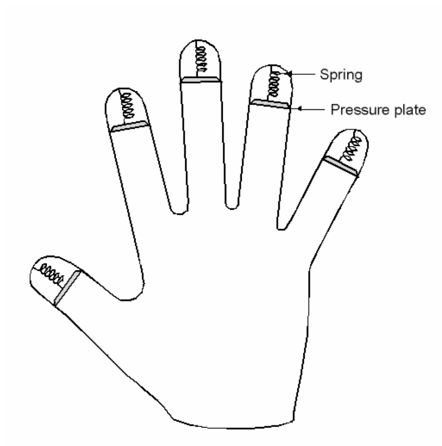
The use of salt to defeat precomputation of encrypted passwords

Authentication Using a Physical Object



- Magnetic cards
  - magnetic stripe cards
  - chip cards: stored value cards, smart cards

Authentication Using Biometrics



A device for measuring finger length.

Countermeasures

- Limiting times when someone can log in
- Automatic callback at number prespecified
- Limited number of login tries
- A database of all logins
- Simple login name/password as a trap
  - security personnel notified when attacker bites

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) Correct login screen
(b) Phony login screen

Logic Bombs

- Company programmer writes program
  - potential to do harm
  - OK as long as he/she enters password daily
  - ff programmer fired, no password and bomb explodes

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);
```

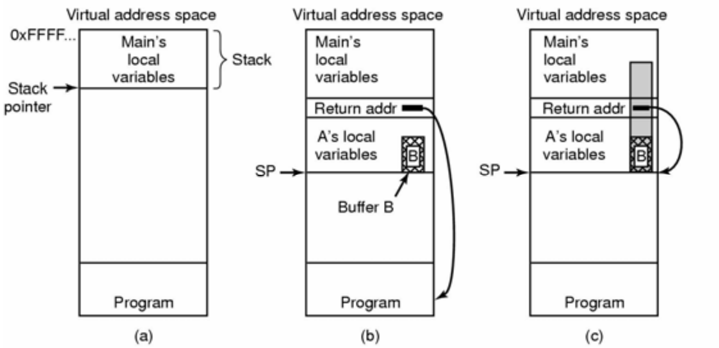
(a)

```
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

Buffer Overflow



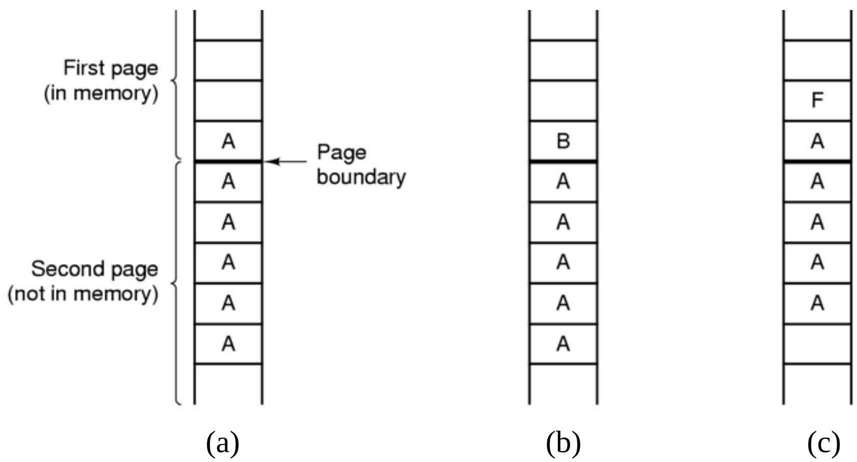
- (a) Situation when main program is running
- (b) After program *A* called
- (c) Buffer overflow shown in gray

Generic Security Attacks

Typical attacks

- Request memory, disk space, tapes and just read
- Try illegal system calls
- Start a login and hit DEL, RUBOUT, or BREAK
- Try modifying complex OS structures
- Try to do specified DO NOTs
- Convince a system programmer to add a trap door
- Beg admin's sec'y to help a poor user who forgot password

Famous Security Flaws



The TENEX – password problem

Design Principles for Security

1. System design should be public
2. Default should be n access
3. Check for current authority
4. Give each process least privilege possible
5. Protection mechanism should be
   - simple
   - uniform
   - in lowest layers of system
6. Scheme should be psychologically acceptable

And … keep it simple

Network Security

- External threat
  - code transmitted to target machine
  - code executed there, doing damage
- Goals of virus writer
  - quickly spreading virus
  - difficult to detect
  - hard to get rid of
- Virus = program can reproduce itself
  - attach its code to another program
  - additionally, do harm

Virus Damage Scenarios

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

How Viruses Work (1)

- Virus written in assembly language
- Inserted into another program
  - use tool called a “dropper”
- Virus dormant until program executed
  - then infects other programs
  - eventually executes its “payload”

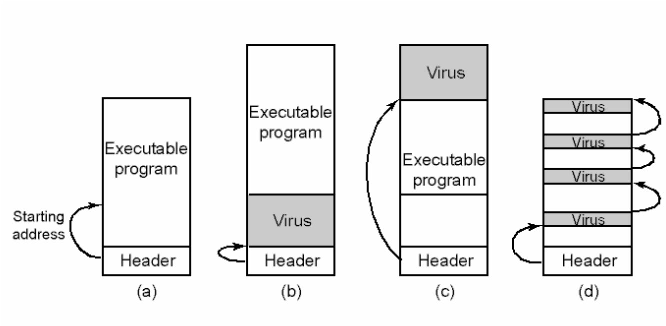
How Viruses Work (2)

Recursive procedure that finds executable files on a UNIX system

Virus could infect them all

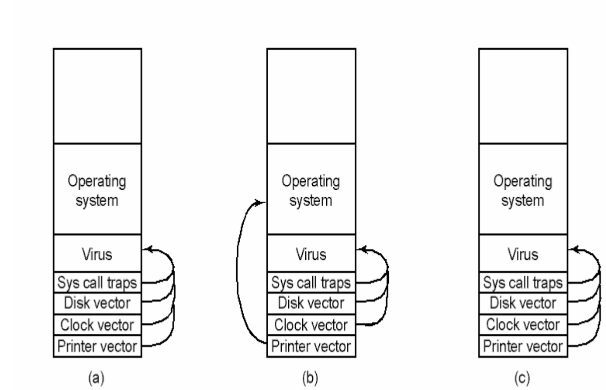
```
#include <sys/types.h>                          /* standard POSIX headers */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf;                               /* for lstat call to see if file is sym link */

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 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

How Viruses Work (4)

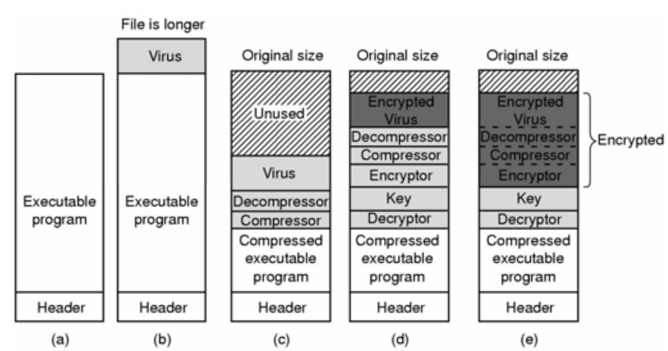


- After virus has captured interrupt, trap vectors
- After OS has retaken printer interrupt vector
- After virus has noticed loss of printer interrupt vector and recaptured it

How Viruses Spread

- Virus placed where likely to be copied
- When copied
  - – infects programs on hard drive, floppy
  - – may try to spread over LAN
- Attach to innocent looking email
  - – when it runs, use mailing list to replicate

Antivirus and Anti-Antivirus Techniques



(a) A program
(b) Infected program
(c) Compressed infected program
(d) Encrypted virus
(e) Compressed virus with encrypted compression code

Antivirus and Anti-Antivirus Techniques

(a)	(b)	(c)	(d)	(e)
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

Examples of a polymorphic virus

All of these examples do the same thing

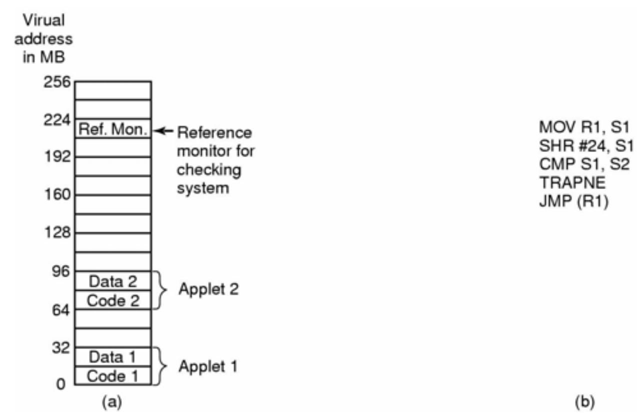
Antivirus and Anti-Antivirus Techniques

- Integrity checkers
- Behavioral checkers
- Virus avoidance
  - good OS
  - install only shrink-wrapped software
  - use antivirus software
  - do not click on attachments to email
  - frequent backups
- Recovery from virus attack
  - halt computer, reboot from safe disk, run antivirus

The Internet Worm

- Consisted of two programs
  - bootstrap to upload worm
  - the worm itself
- Worm first hid its existence
- Next replicated itself on new machines

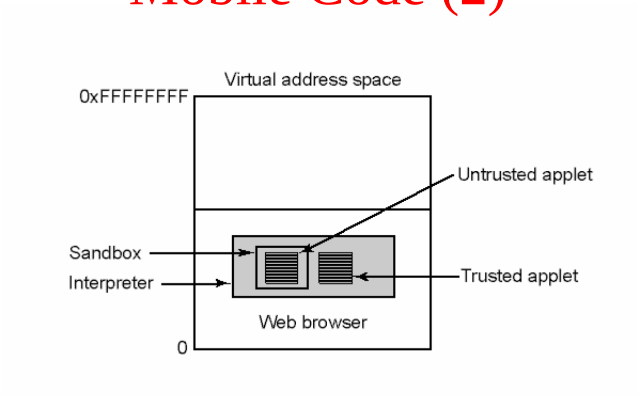
Mobile Code (1) Sandboxing



(a) Memory divided into 1-MB sandboxes

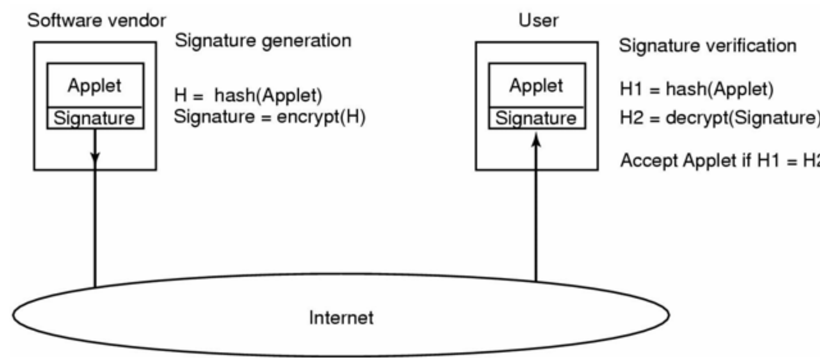
(b) One way of checking an instruction for validity

Mobile Code (2)



Applets can be interpreted by a Web browser

Mobile Code (3)



How code signing works

Java Security (1)

- A type safe language
  - compiler rejects attempts to misuse variable
- Checks include …
  1. Attempts to forge pointers
  2. Violation of access restrictions on private class members
  3. Misuse of variables by type
  4. Generation of stack over/underflows
  5. Illegal conversion of variables to another type

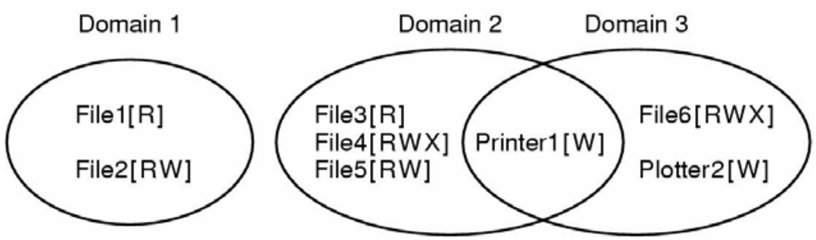
Java Security (2)

URL	Signer	Object	Action
www.taxprep.com	TaxPrep	/usr/susan/1040.xls	Read
*		/usr/tmp/*	Read, Write
www.microsoft.com	Microsoft	/usr/susan/Office/–	Read, Write, Delete

Examples of specified protection with JDK 1.2

Protection Mechanisms

Protection Domains (1)



Examples of three protection domains

Protection Domains (2)

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

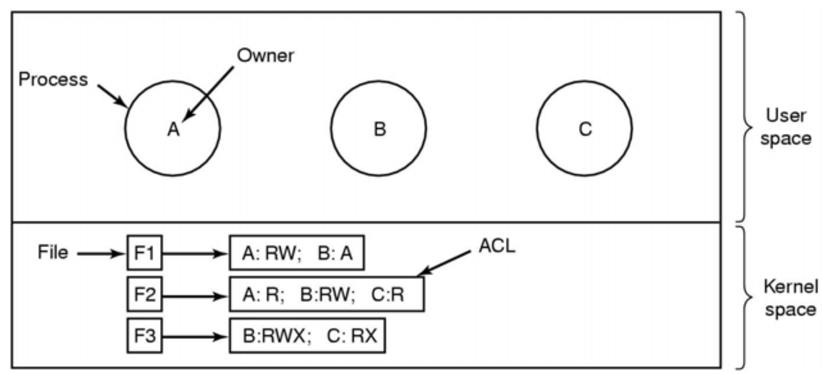
A protection matrix

Protection Domains (3)

main	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
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

Access Control Lists (1)



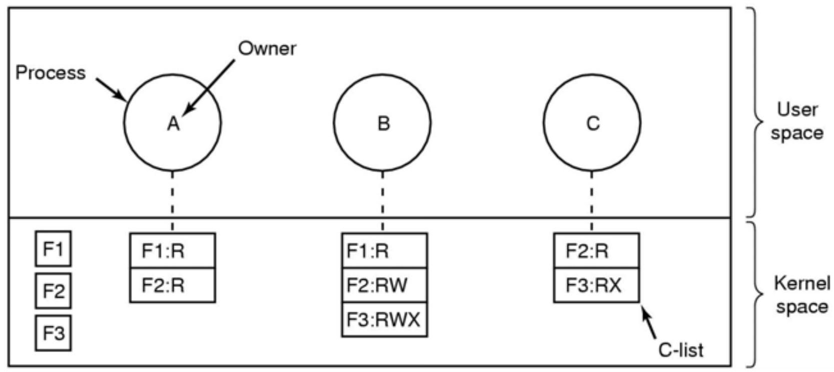
Use of access control lists of manage file access

Access Control Lists (2)

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

Two access control lists

Capabilities (1)



Each process has a capability list

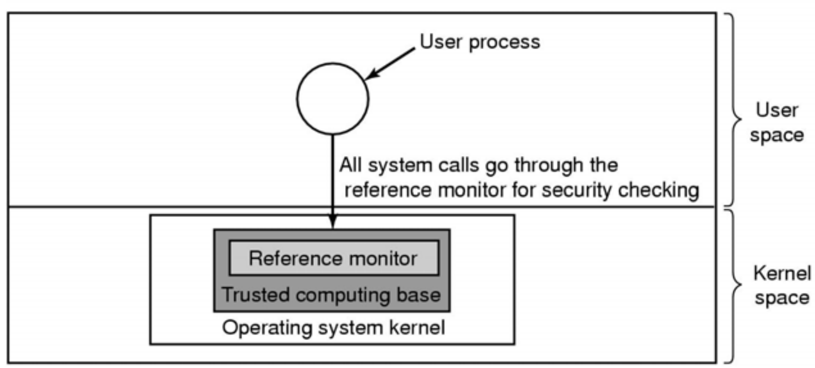
Capabilities (2)

- Cryptographically-protected capability

Server	Object	Rights	f(Objects, Rights, Check)

- Generic Rights
  1. Copy capability
  2. Copy object
  3. Remove capability
  4. Destroy object

Trusted Systems
Trusted Computing Base



A reference monitor

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

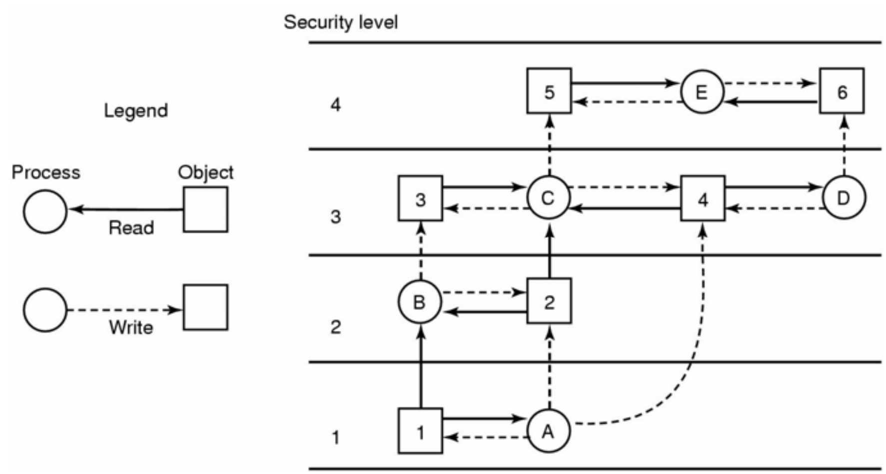
	Compiler	Mailbox 7	Secret
Eric	Read Execute		
Henry	Read Execute	Read Write	
Robert	Read Execute	Read	Read Write

(b)

(a) An authorized state

(b) An unauthorized state

Multilevel Security (1)



The Bell-La Padula multilevel security model

Multilevel Security (2)

The Biba Model

- Principles to guarantee integrity of data

1. Simple integrity principle
   - process can write only objects at its security level or lower
2. The integrity * property
   - process can read only objects at its security level or higher

Orange Book Security (1)

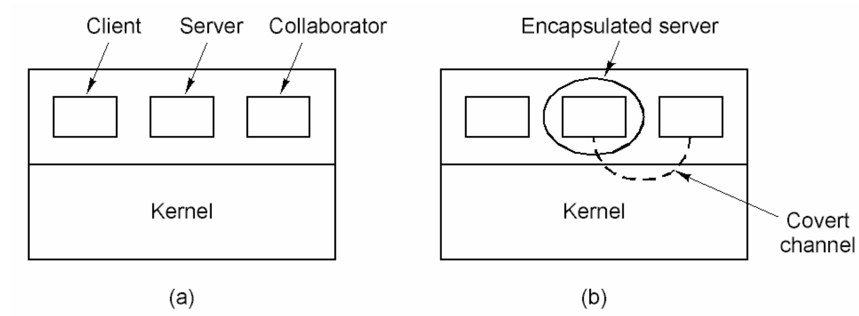
Criterion	D	C1	C2	B1	B2	B3	A1
Security policy							
Discretionary access control		X	X	→	→	X	→
Object reuse			X	→	→	→	→
Labels				X	X	→	→
Label integrity				X	→	→	→
Exportation of labeled information				X	→	→	→
Labeling human readable output				X	→	→	→
Mandatory access control				X	X	→	→
Subject sensitivity 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

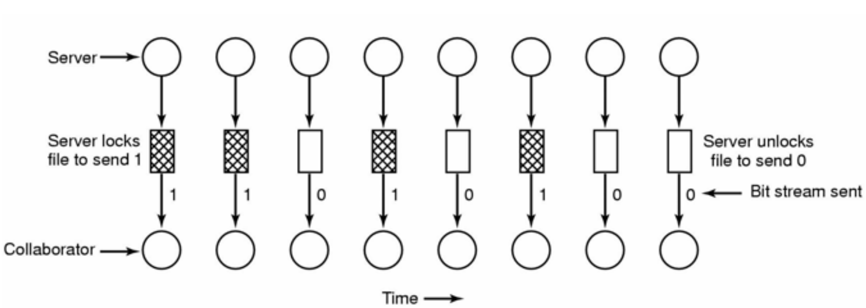
Covert Channels (1)



Client, server and collaborator processes

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