MODERN OPERATING SYSTEMS

Third Edition

ANDREW S. TANENBAUM

Chapter 9
Security

Threats

Goal	Threat
Data confidentiality	Exposure of data
Data integrity	Tampering with data
System availability	Denial of service
Exclusion of outsiders	System takeover by viruses

Figure 9-1. Security goals and threats.

Intruders

Common categories:

- Casual prying by nontechnical users.
- Snooping by insiders.
- Determined attempts to make money.
- Commercial or military espionage.

Accidental Data Loss

Common causes of accidental data loss:

- Acts of God: fires, floods, earthquakes, wars, riots, or rats gnawing backup tapes.
- Hardware or software errors: CPU malfunctions, unreadable disks or tapes, telecommunication errors, program bugs.
- Human errors: incorrect data entry, wrong tape or CD-ROM mounted, wrong program run, lost disk or tape, or some other mistake.

Basics Of Cryptography

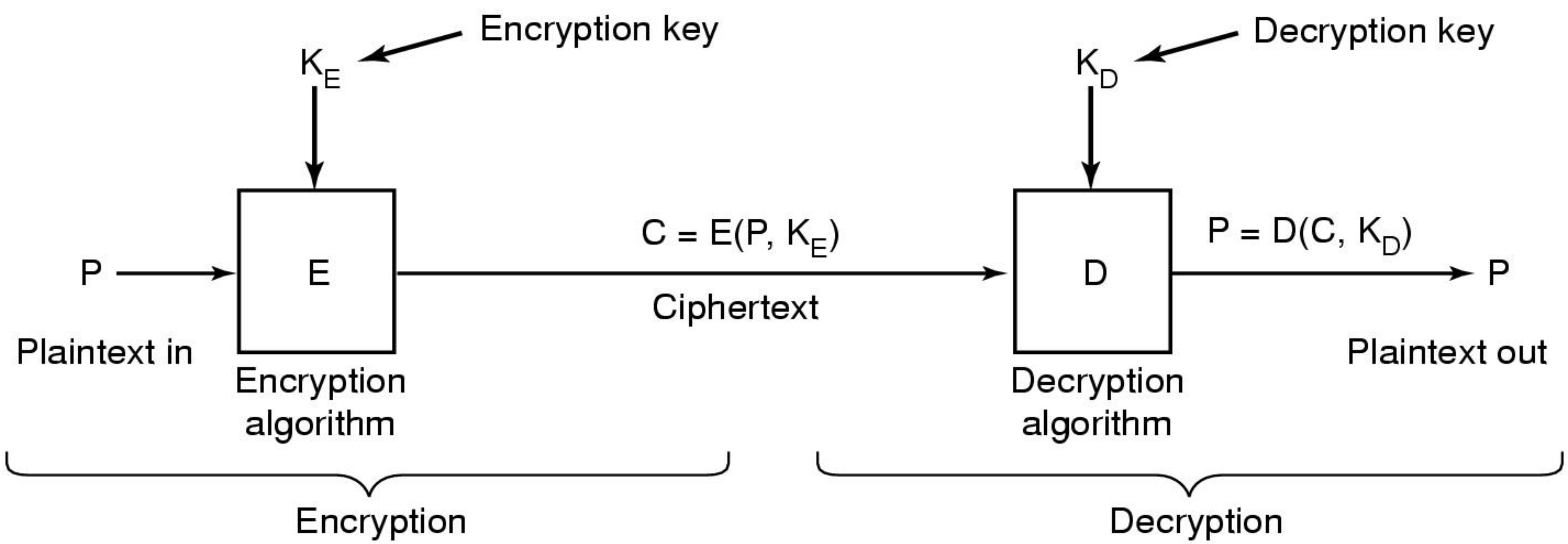


Figure 9-2. Relationship between the plaintext and the ciphertext.

Secret-Key Cryptography

Monoalphabetic substitution:

Plaintext: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`

Ciphertext: `QWERTYUIOPASDFGHJKLZXCVBNM`

Public-Key Cryptography

- Encryption makes use of an "easy" operation, such as how much is
  314159265358979 × 314159265358979?

- Decryption without the key requires you to perform a hard operation, such as what is the square root of
  3912571506419387090594828508241?

Digital Signatures

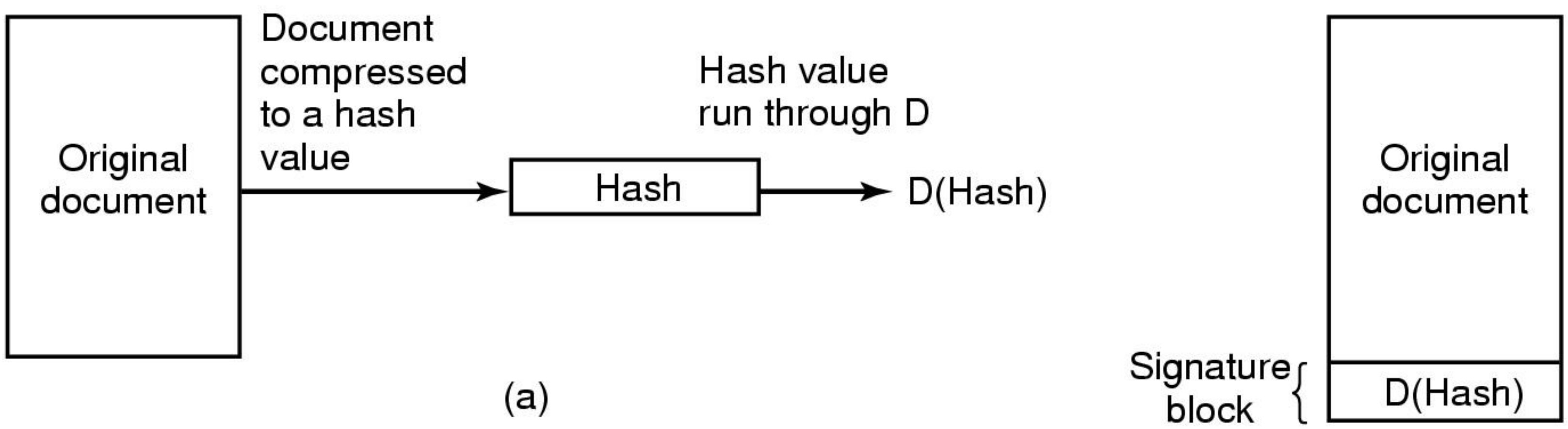


Figure 9-3. (a) Computing a signature block.
(b) What the receiver gets.

Protection Domains (1)

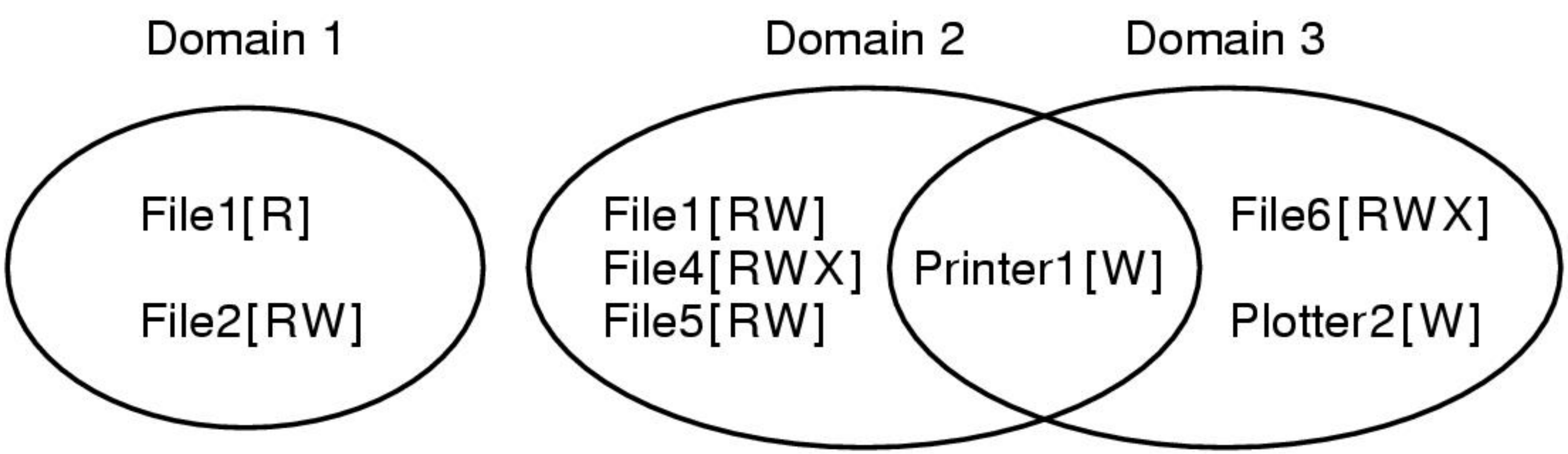


Figure 9-4. Three protection domains.

Protection Domains (2)

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

Figure 9-5. A protection matrix.

Protection Domains (3)

Domain	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			

Figure 9-6. A protection matrix with domains as objects.

Access Control Lists (1)

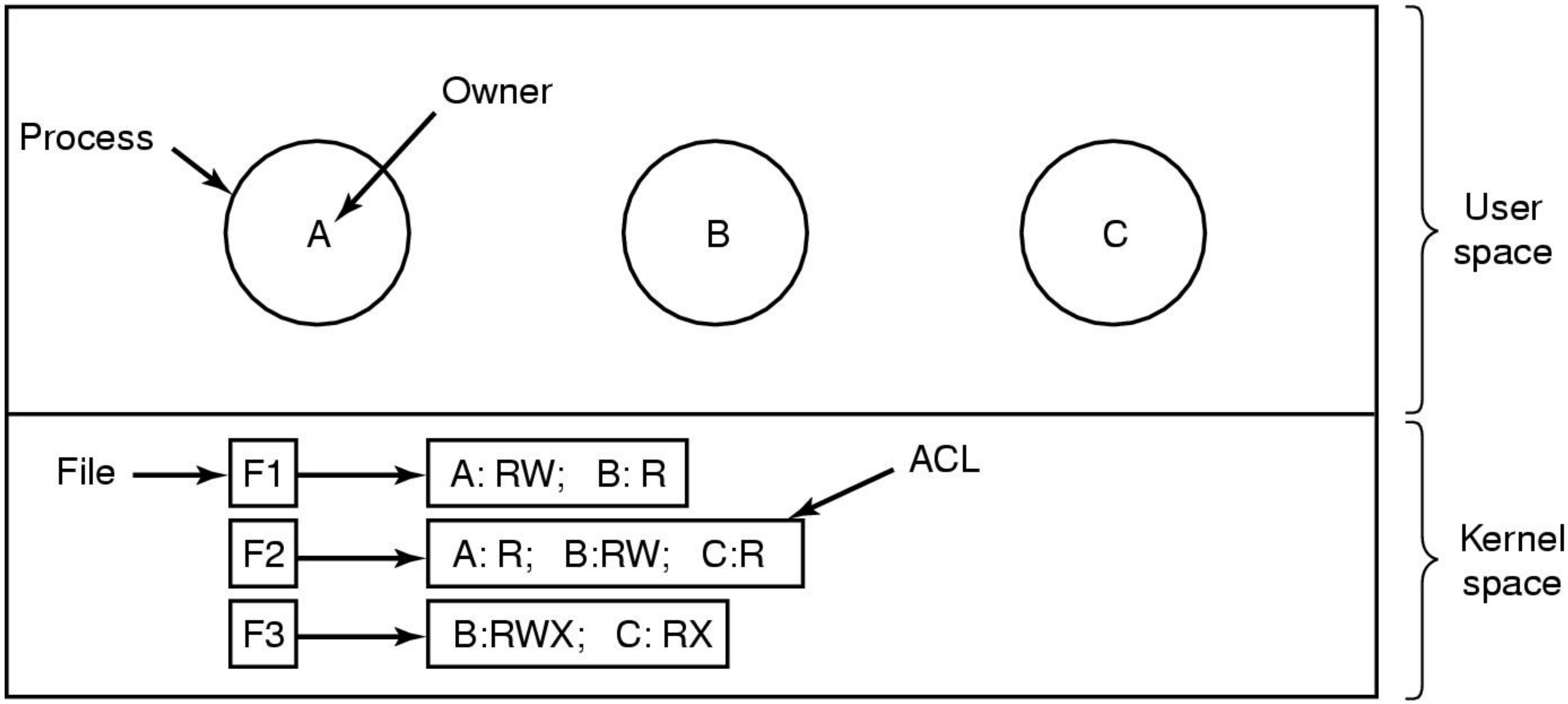


Figure 9-7. Use of access control lists to manage file access.

Access Control Lists (2)

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

Figure 9-8. Two access control lists.

Capabilities (1)

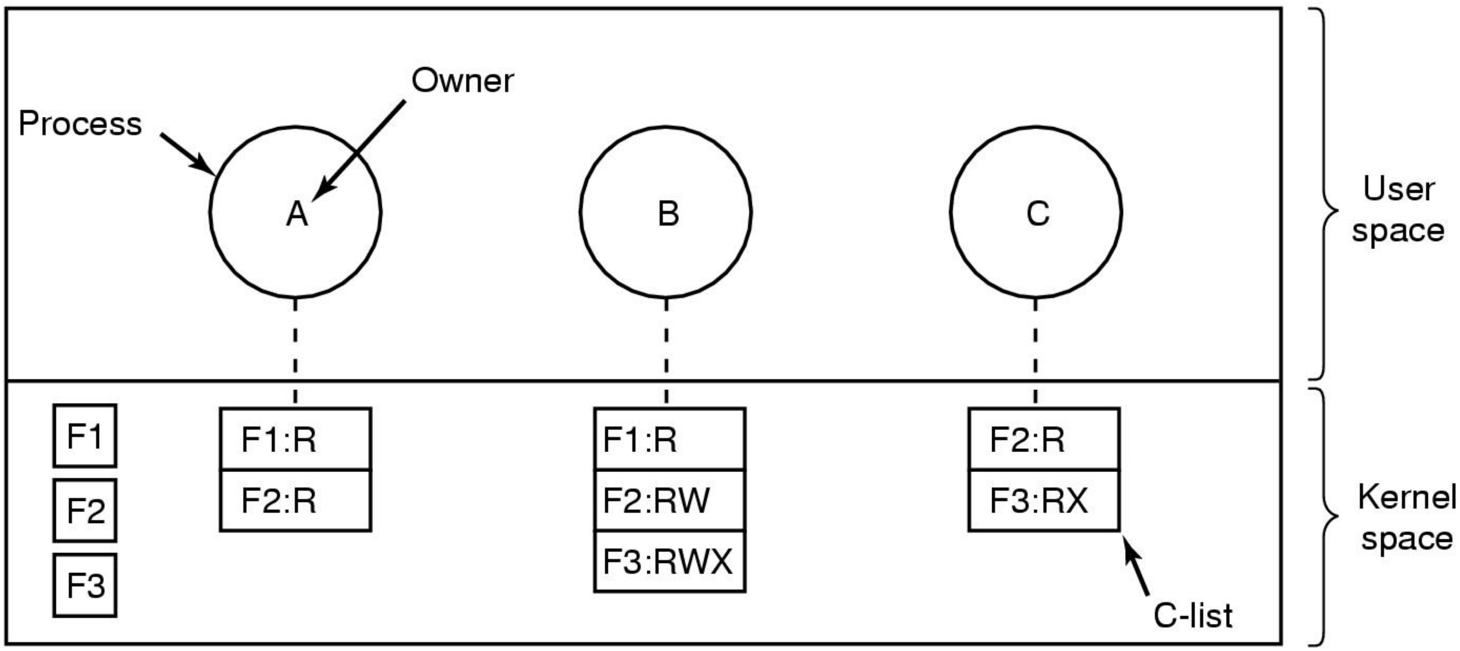


Figure 9-9. When capabilities are used, each process has a capability list.

Capabilities (2)

Server	Object	Rights	f(Objects,Rights,Check)

Figure 9-10. A cryptographically protected capability.

Capabilities (3)

Examples of generic rights:

- Copy capability: create a new capability for the same object.
- Copy object: create a duplicate object with a new capability.
- Remove capability: delete an entry from the C-list; object unaffected.
- Destroy object: permanently remove an object and a capability.

Trusted Systems

- Consider reports of viruses, worms, etc.
- Two naive (but logical) questions:
  - Is it possible to build a secure computer system?
  - If so, why is it not done?

Trusted Computing Base

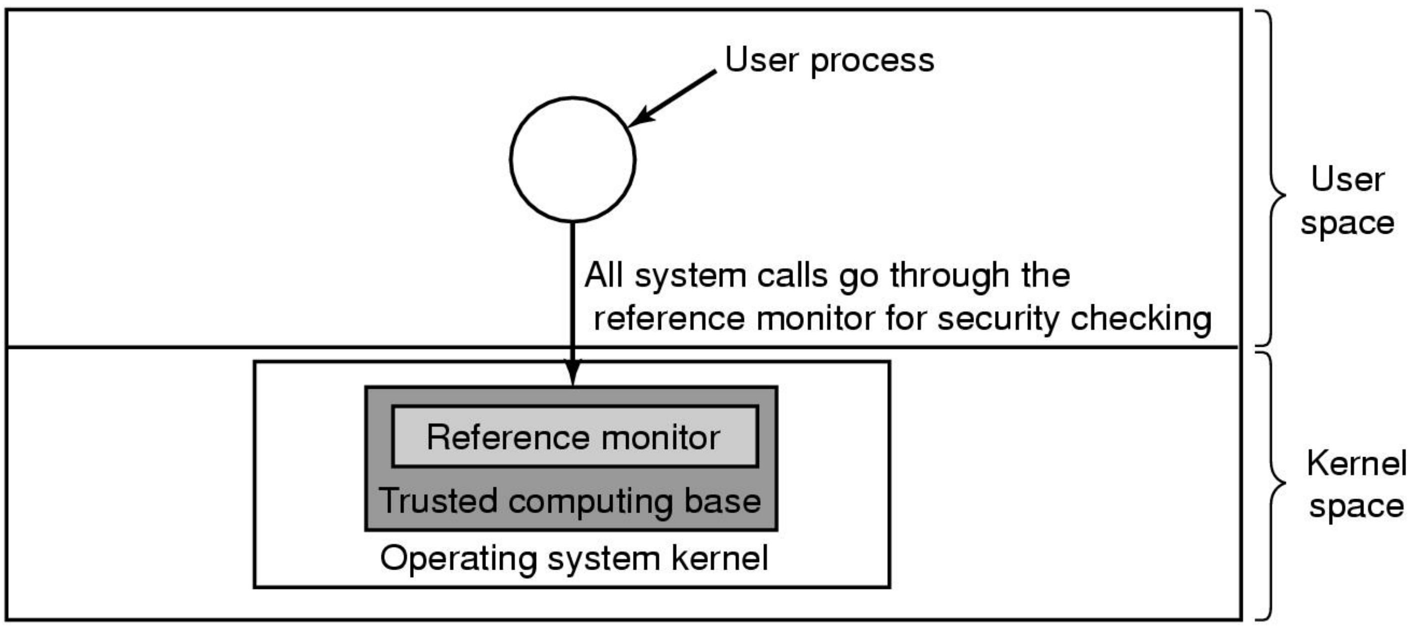


Figure 9-11. A reference monitor.

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)

Figure 9-12. (a) An authorized state. (b) An unauthorized state.

The Bell-La Padula Model (1)

Rules for the Bell-La Padula model:

- **The simple security property**: A process running at security level k can read only objects at its level or lower.
- **The * property**: A process running at security level k can write only objects at its level or higher.

The Bell-La Padula Model (2)

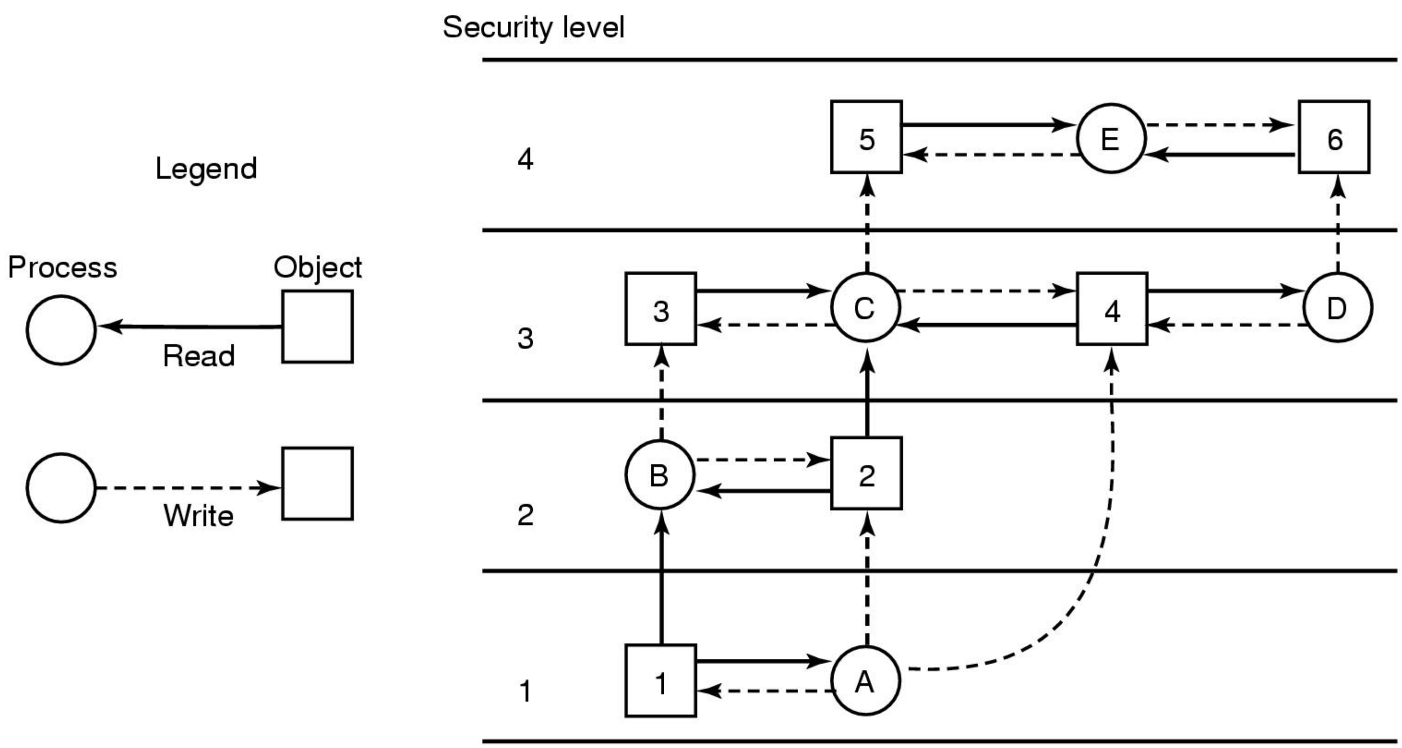


Figure 9-13. The Bell-La Padula multilevel security model.

The Biba Model

Rules for the Biba model:

- **The simple integrity principle**: A process running at security level k can write only objects at its level or lower (no write up).
- **The integrity * property**: A process running at security level k can read only objects at its level or higher (no read down).

Covert Channels (1)

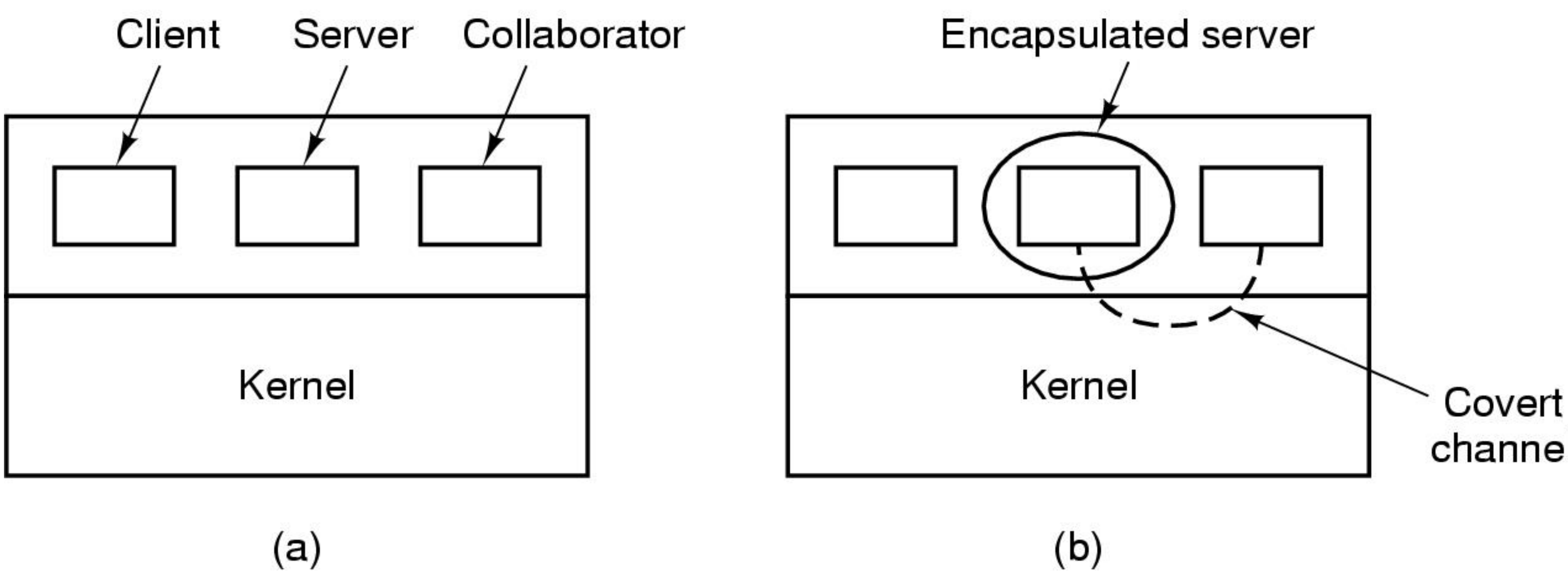


Figure 9-14. (a) The client, server, and collaborator processes. (b) The encapsulated server can still leak to the collaborator via covert channels.

Covert Channels (2)

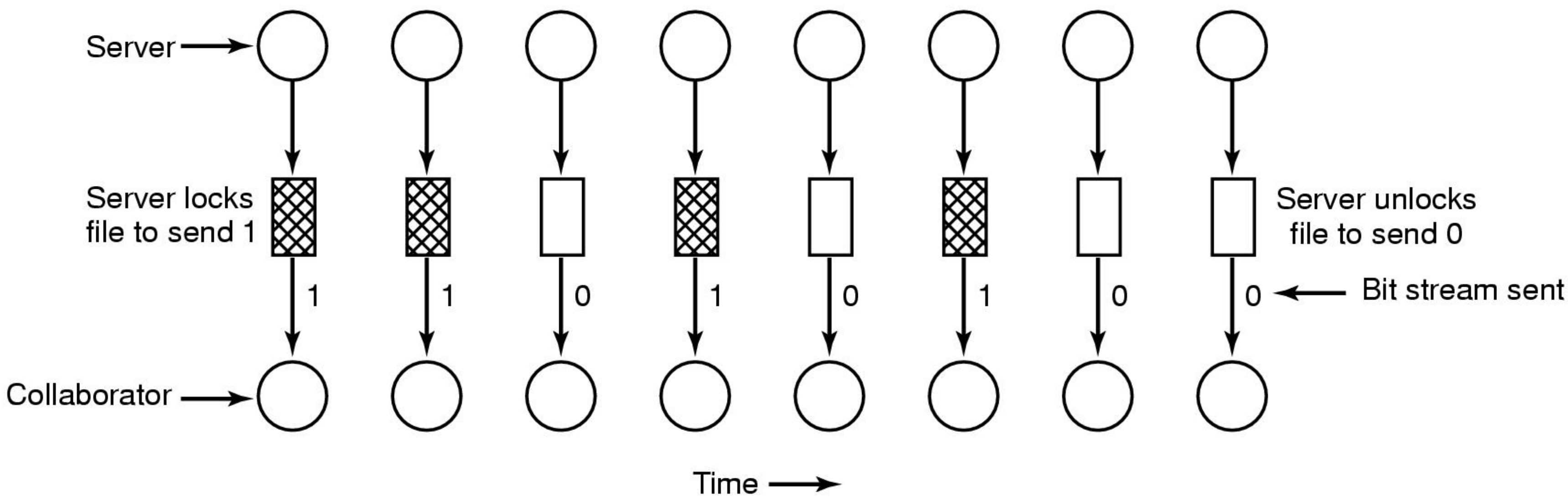
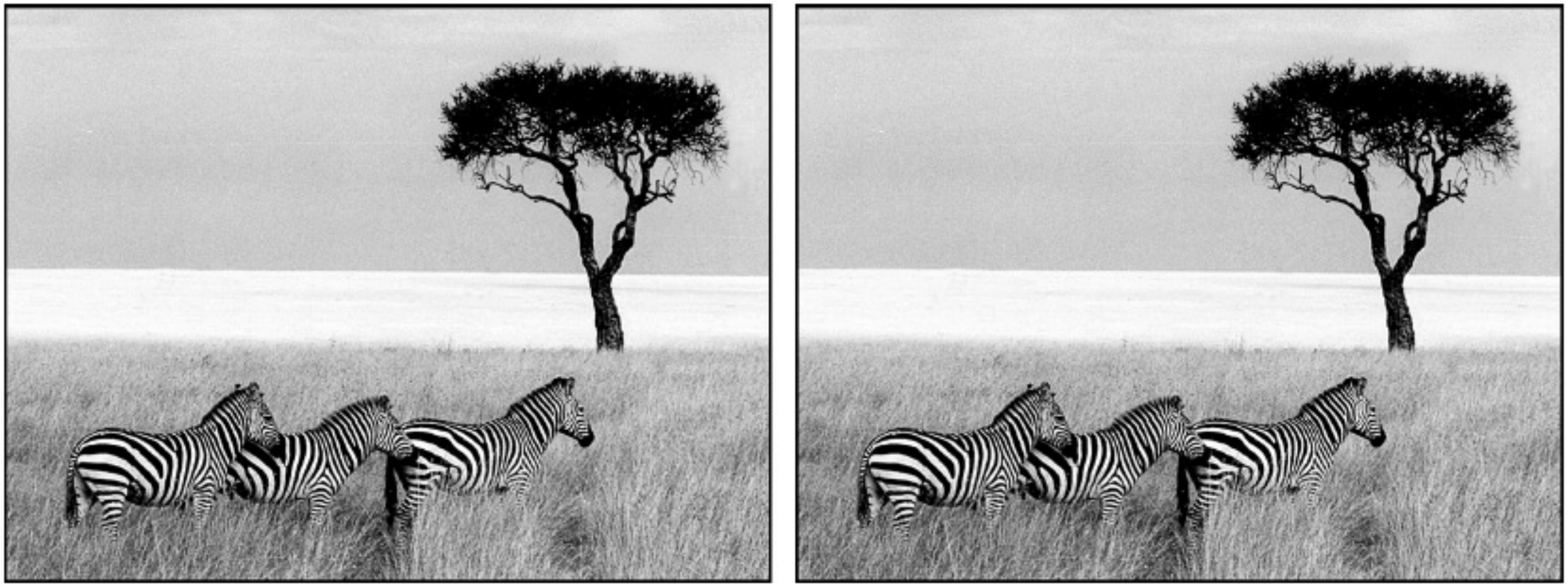


Figure 9-15. A covert channel using file locking.

Covert Channels (3)



(a) (b)

Figure 9-16. (a) Three zebras and a tree. (b) Three zebras, a tree, and the complete text of five plays by William Shakespeare.

Authentication

General principles of authenticating users:

- Something the user knows.
- Something the user has.
- Something the user is.

Authentication Using Passwords

```
LOGIN: mitch
PASSWORD: FooBar!-7
SUCCESSFUL LOGIN
```

(a)

```
LOGIN: carol
INVALID LOGIN NAME
LOGIN:
```

(b)

```
LOGIN: carol
PASSWORD: Idunno
INVALID LOGIN
LOGIN:
```

(c)

Figure 9-17. (a) A successful login.
(b) Login rejected after name is entered.
(c) Login rejected after name and password are typed.

How Crackers Break In

```
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
```

Figure 9-18. How a cracker broke into a U.S. Department of Energy computer at LBL.

UNIX Password Security

Bobbie, 4238, e(Dog, 4238)
Tony, 2918, e(6%%TaeFF, 2918)
Laura, 6902, e(Shakespeare, 6902)
Mark, 1694, e(XaB#Bwcz, 1694)
Deborah, 1092, e(LordByron,1092)

Figure 9-19. The use of salt to defeat precomputation of encrypted passwords.

Challenge-Response Authentication

The questions should be chosen so that the user does not need to write them down.

Examples:

- Who is Marjolein's sister?
- On what street was your elementary school?
- What did Mrs. Woroboff teach?

Authentication Using a Physical Object

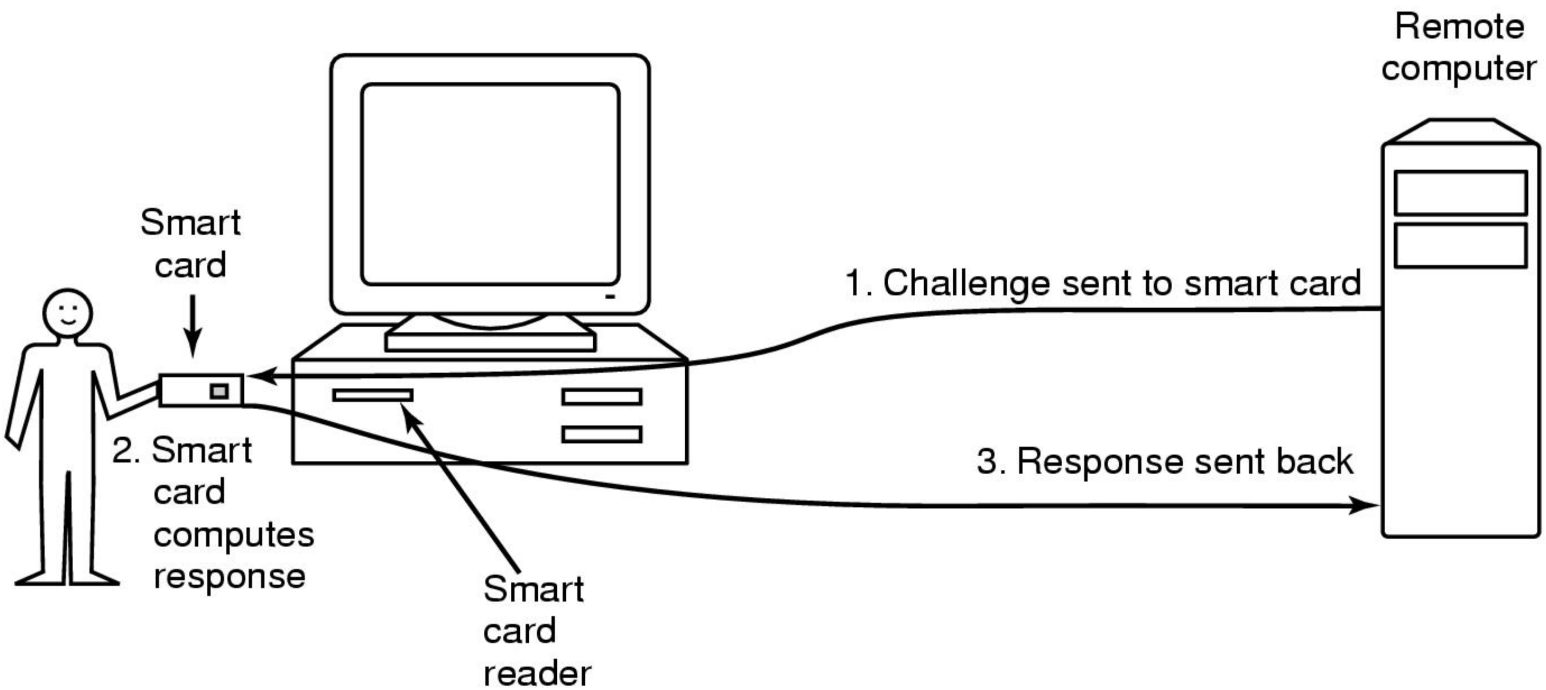


Figure 9-20. Use of a smart card for authentication.

Authentication Using Biometrics

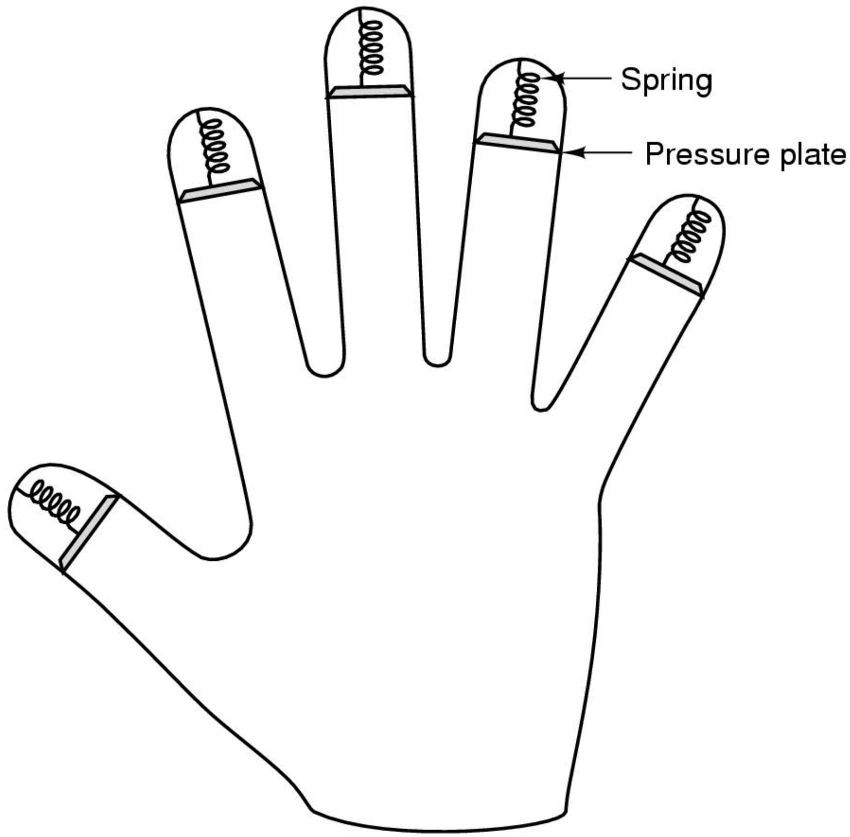


Figure 9-21. A device for measuring finger length.

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)

Figure 9-22. (a) Normal code. (b) Code with a trap door inserted.

Login Spoofing

(a) (b)

Figure 9-23. (a) Correct login screen. (b) Phony login screen.

Exploiting Code Bugs

Example steps to exploit a bug:

- Run port scan to find machines that accept telnet connections.
- Try to log in by guessing login name and password combinations.
- Once in, run the flawed program with input that triggers the bug.
- If the buggy program is SETUID root, create a SETUID root shell.
- Fetch and start a zombie program that listens to an IP port for cmds.
- Arrange that the zombie program is started when the system reboots.

Buffer Overflow Attacks

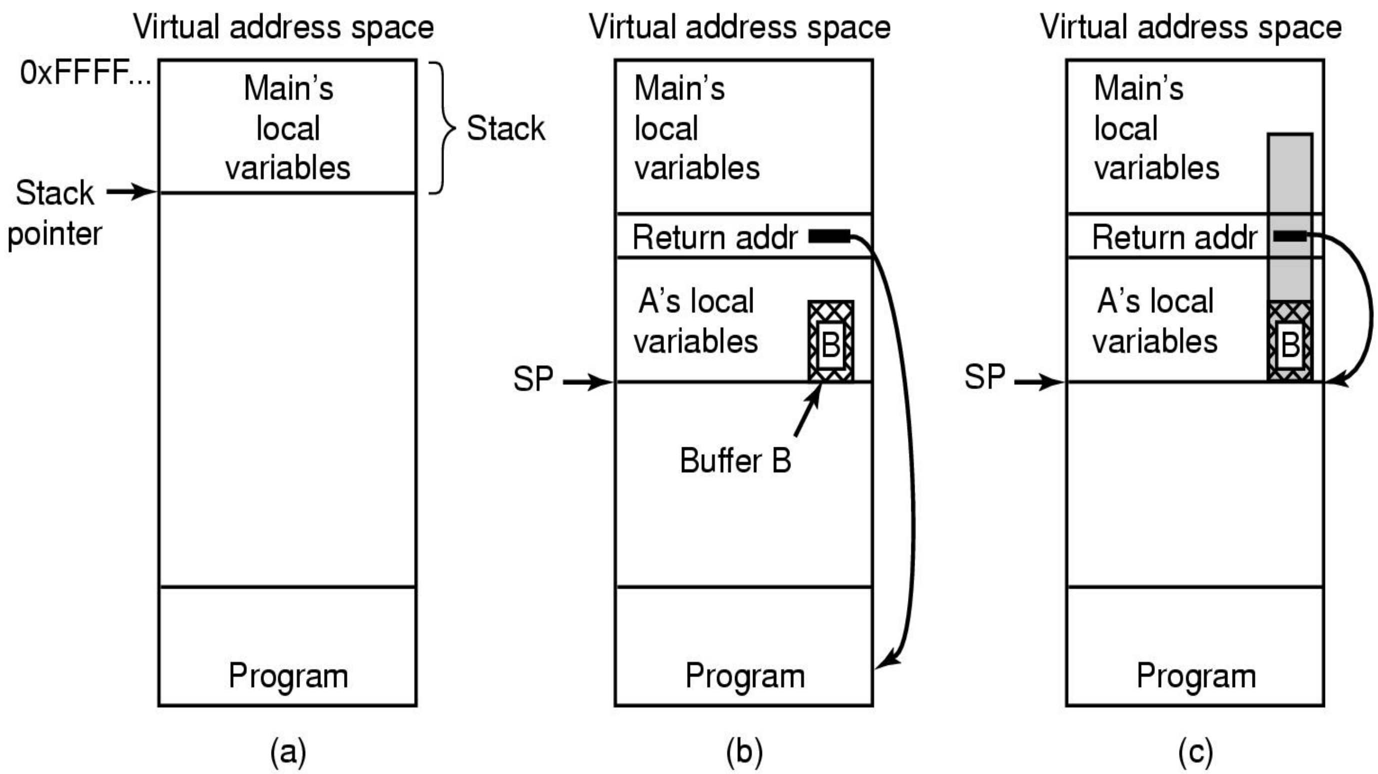


Figure 9-24. (a) Situation when the main program is running. (b) After the procedure A has been called. (c) Buffer overflow shown in gray.

Return to libc Attacks

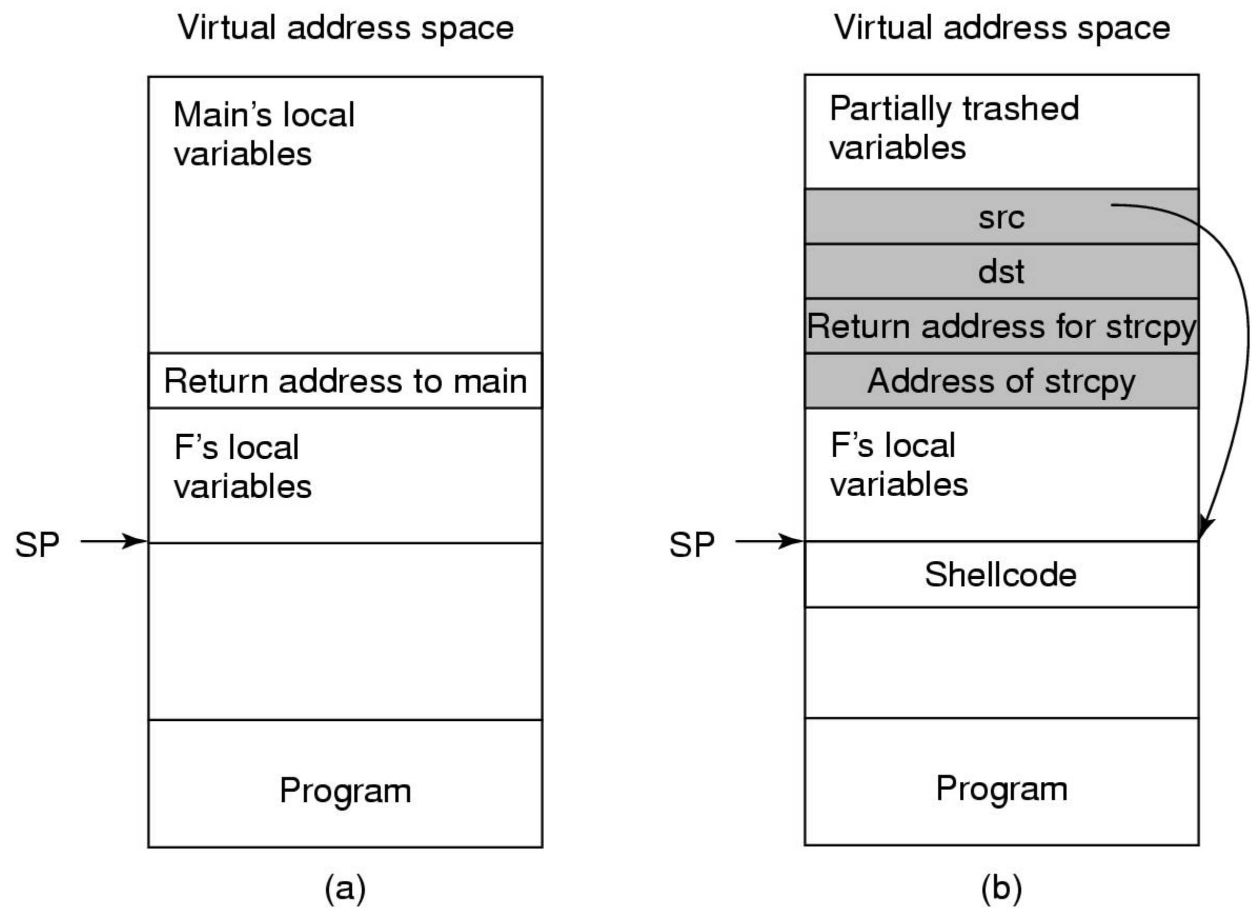


Figure 9-25. (a) The stack before the attack.
(b) The stack after the stack has been overwritten.

Code Injection Attacks

```
int main(int argc, char *argv[])
{
  char src[100], dst[100], cmd[205] = "cp ";              /* declare 3 strings */
  printf("Please enter name of source file: ");           /* ask for source file */
  gets(src);                                              /* get input from the keyboard */
  strcat(cmd, src);                                       /* concatenate src after cp */
  strcat(cmd, " ");                                       /* add a space to the end of cmd */
  printf("Please enter name of destination file: ");      /* ask for output file name */
  gets(dst);                                              /* get input from the keyboard */
  strcat(cmd, dst);                                       /* complete the commands string */
  system(cmd);                                            /* execute the cp command */
}
```

Figure 9-26. Code that might lead to a code injection attack.

Malware

Can be used for a form of blackmail.

Example: Encrypts files on victim disk, then displays message …

Greetings from General Encryption

To purchase a decryption key for your hard disk, please send $100 in small unmarked bills to Box 2154, Panama City, Panama.
Thank you. We appreciate your business.

Types of Viruses

- Companion virus
- Executable program virus
- Parasitic virus
- Memory-resident virus
- Boot sector virus
- Device driver virus
- Macro virus
- Source code virus

Executable Program Viruses (1)

```
#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 */
  . . .
```

Figure 9-27. A recursive procedure that finds executable files on a UNIX system.

Executable Program Viruses (2)

```
. . .
    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 */
}
```

Figure 9-27. A recursive procedure that finds executable files on a UNIX system.

Parasitic Viruses

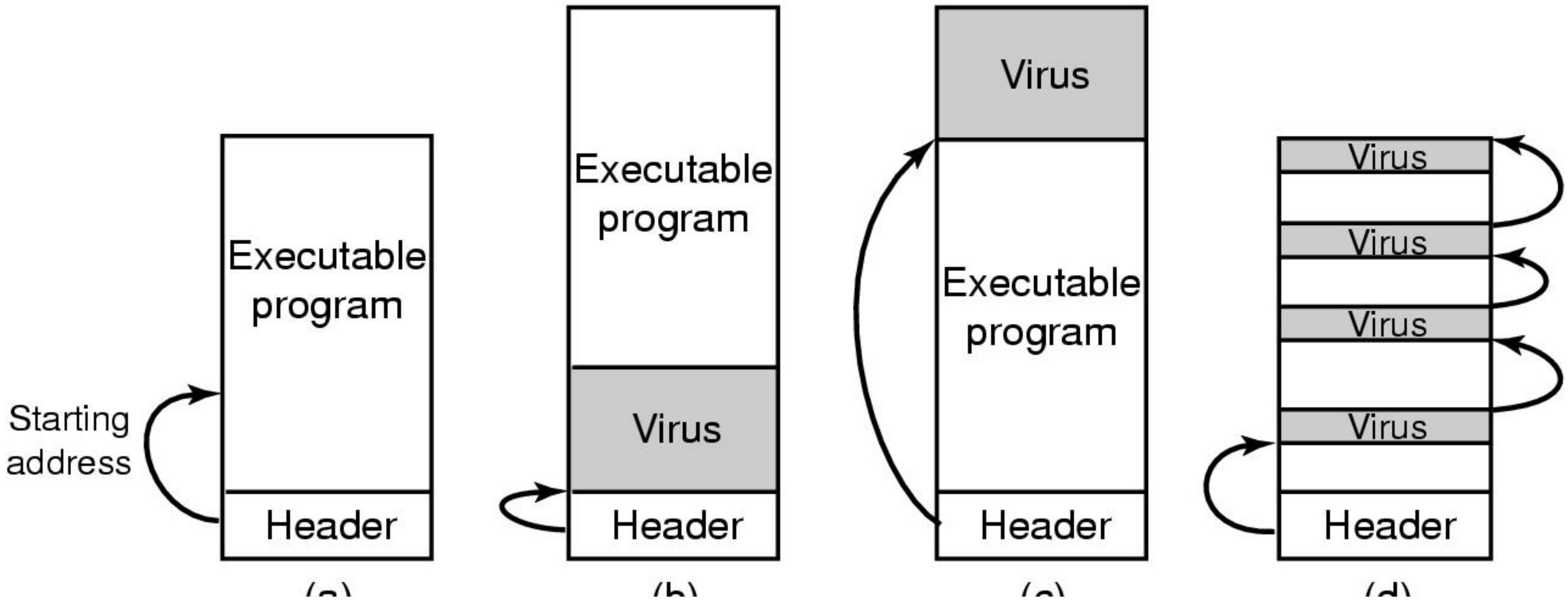


Figure 9-28. (a) An executable program. (b) With a virus at the front. (c) With a virus at the end. (d) With a virus spread over free space within the program.

Boot Sector Viruses

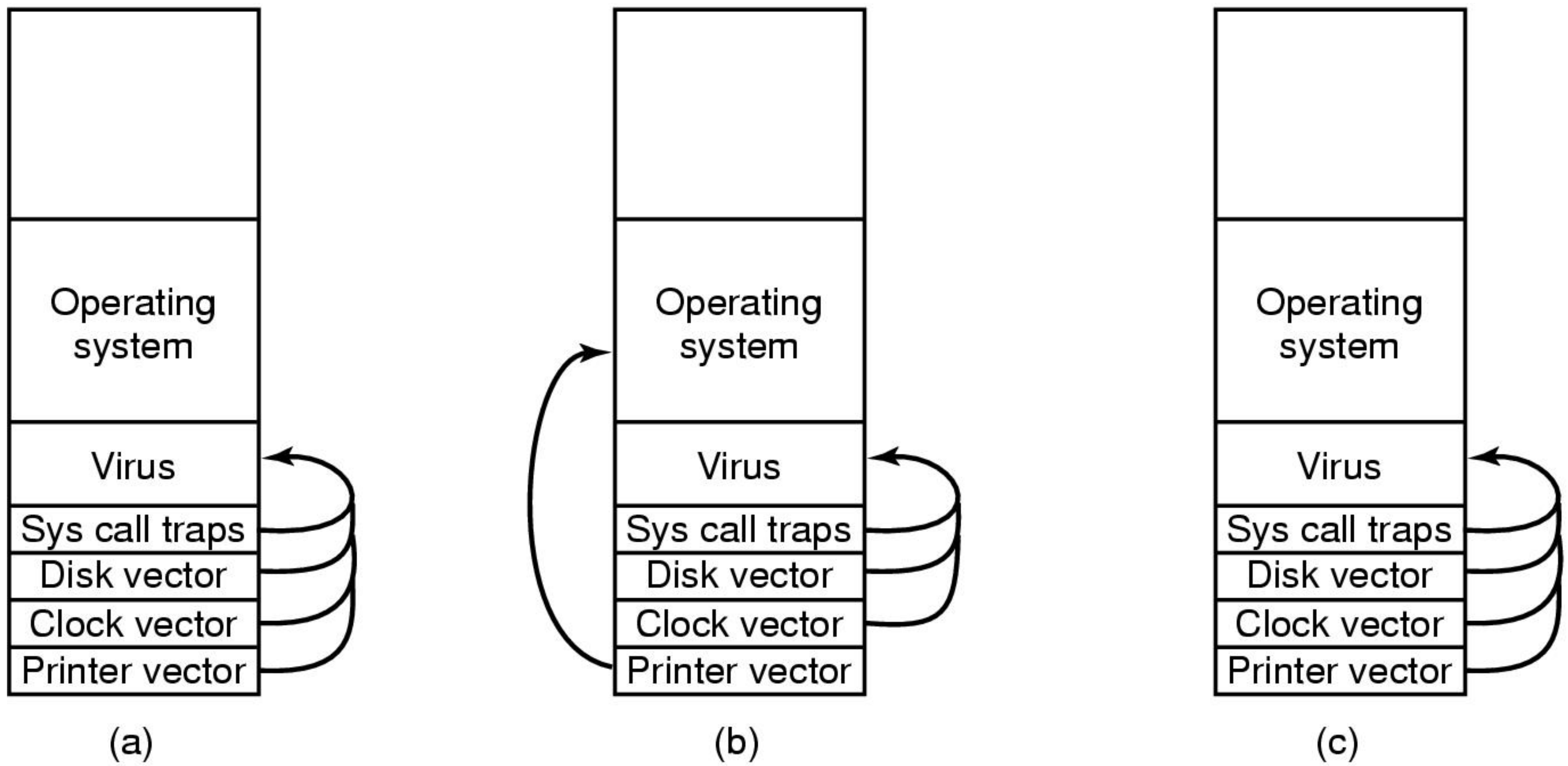


Figure 9-29. (a) After the virus has captured all the interrupt and trap vectors. (b) After the operating system has retaken the printer interrupt vector. (c) After the virus has noticed the loss of the printer interrupt vector and recaptured it.

Spyware (1)

Description:

- Surreptitiously loaded onto a PC without the owner's knowledge
- Runs in the background doing things behind the owner's back

Spyware (2)

Characteristics:

- Hides, victim cannot easily find
- Collects data about the user
- Communicates the collected information back to its distant master
- Tries to survive determined attempts to remove it

How Spyware Spreads

Possible ways:

- Same as malware, Trojan horse
- Drive-by download, visit an infected web site
  - Web pages tries to run an .exe file
  - Unsuspecting user installs an infected toolbar
  - Malicious activeX controls get installed

Actions Taken by Spyware

- Change the browser's home page.
- Modify the browser's list of favorite (bookmarked) pages.
- Add new toolbars to the browser.
- Change the user's default media player.
- Change the user's default search engine.
- Add new icons to the Windows desktop.
- Replace banner ads on Web pages with those the spyware picks.
- Put ads in the standard Windows dialog boxes
- Generate a continuous and unstoppable stream of pop-up ads.

Types of Rootkits (1)

- Firmware rootkits
- Hypervisor rootkits
- Kernel rootkits
- Library rootkits
- Application rootkits

Types of Rootkits (2)

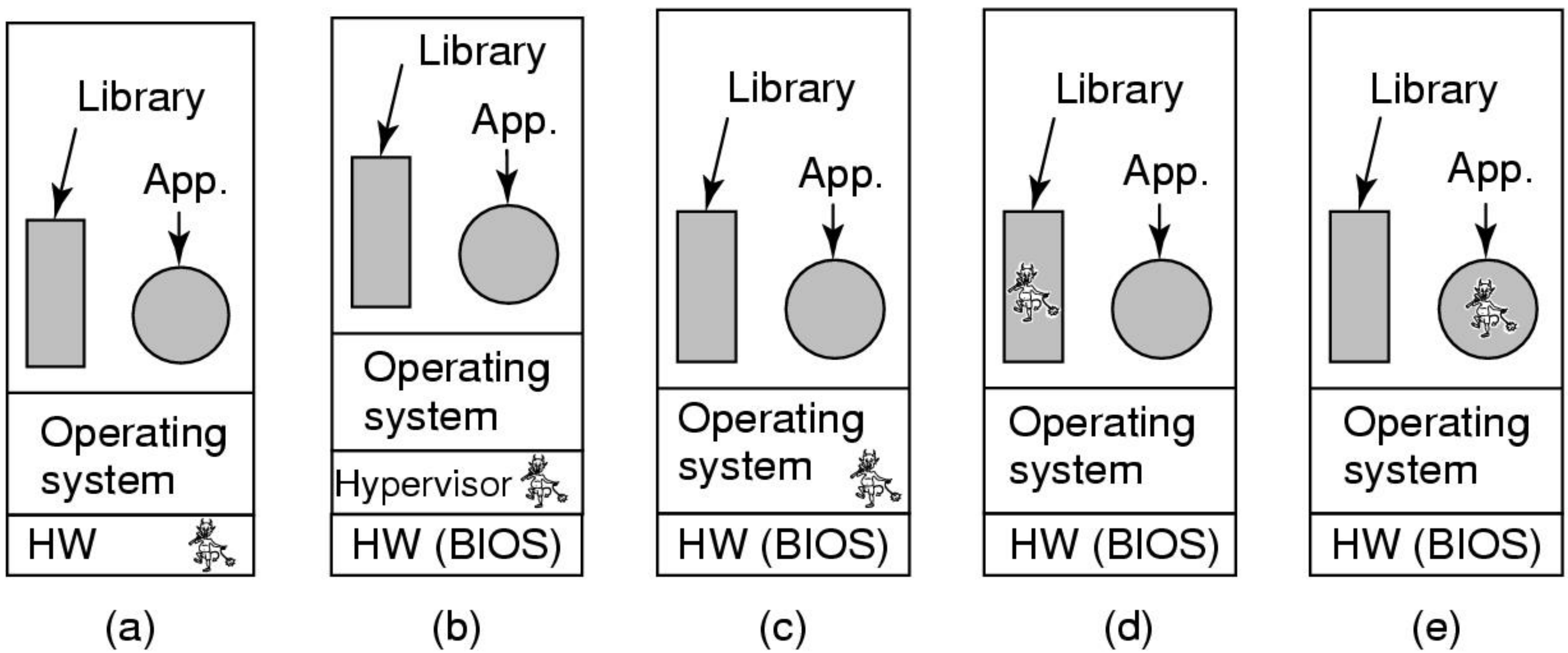


Figure 9-30. Five places a rootkit can hide.

Firewalls

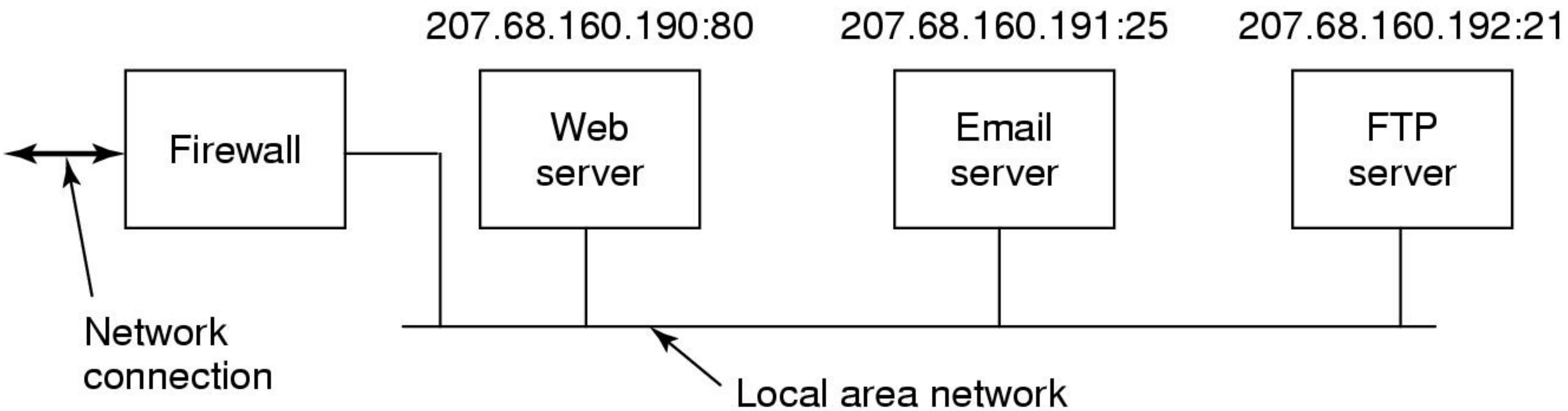


Figure 9-31. A simplified view of a hardware firewall protecting a LAN with three computers.

Virus Scanners (1)

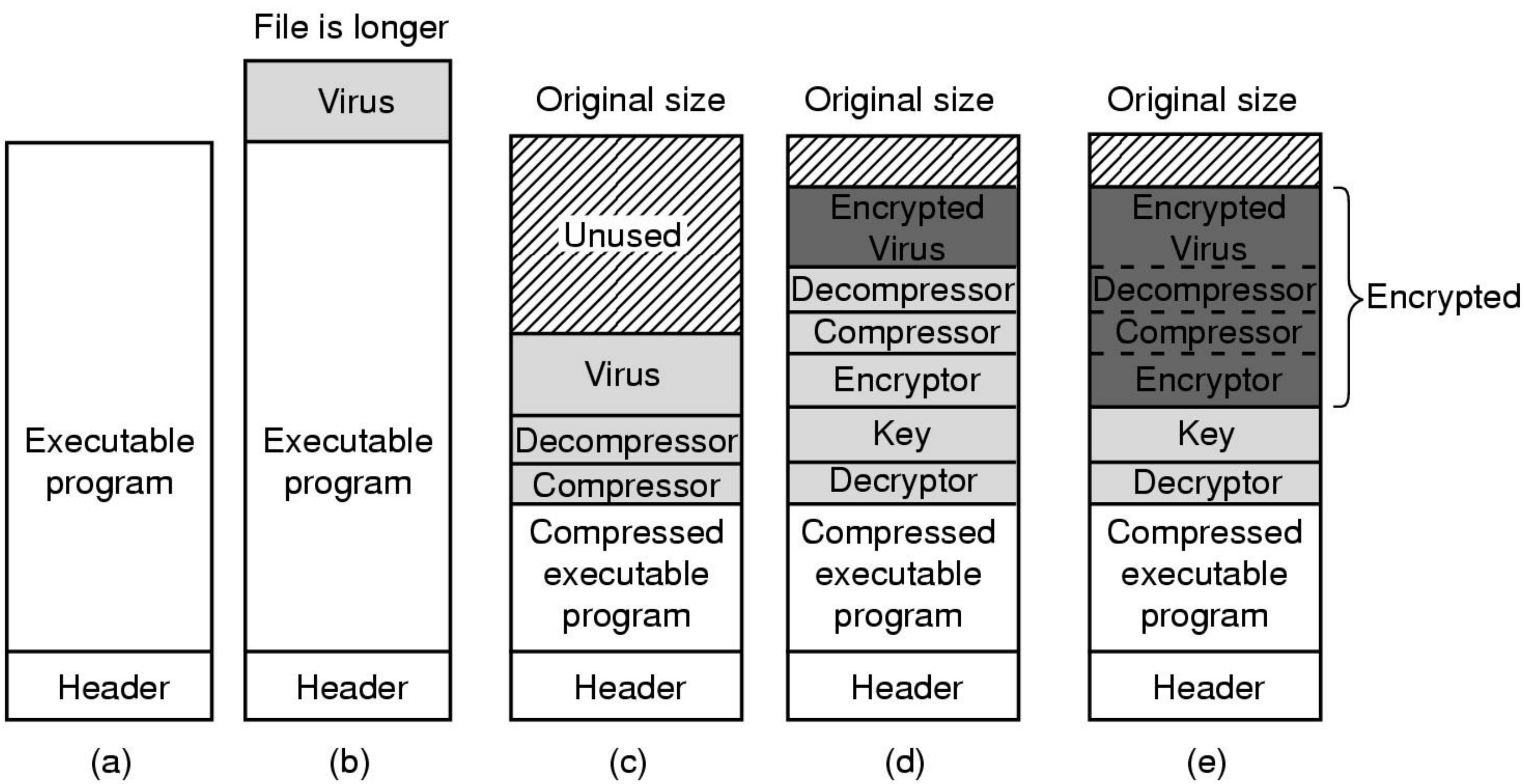


Figure 9-32. (a) A program. (b) An infected program. (c) A compressed infected program. (d) An encrypted virus. (e) A compressed virus with encrypted compression code.

Virus Scanners (2)

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)

Figure 9-33. Examples of a polymorphic virus.

Antivirus and Anti-Antivirus Techniques

- Virus scanners
- Integrity checkers
- Behavioral checkers
- Virus avoidance

Code Signing

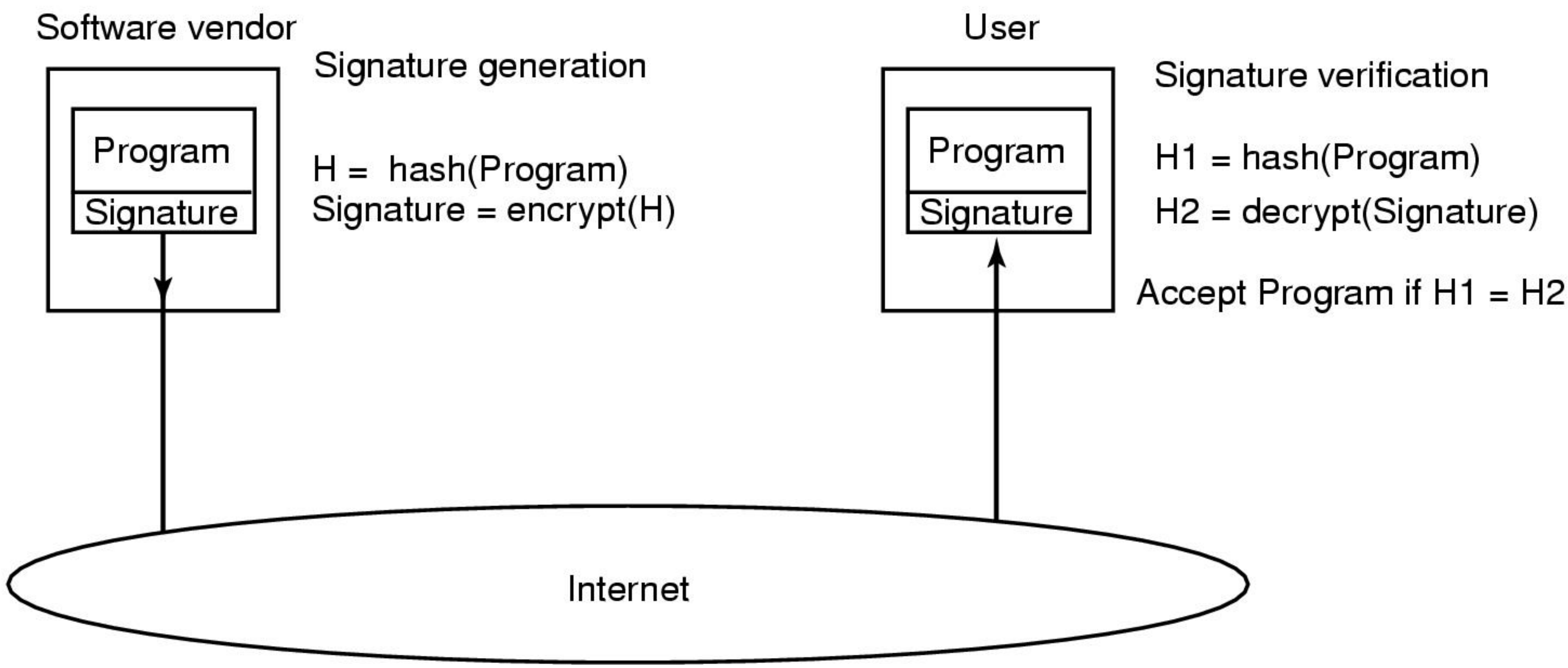


Figure 9-34. How code signing works.

Jailing

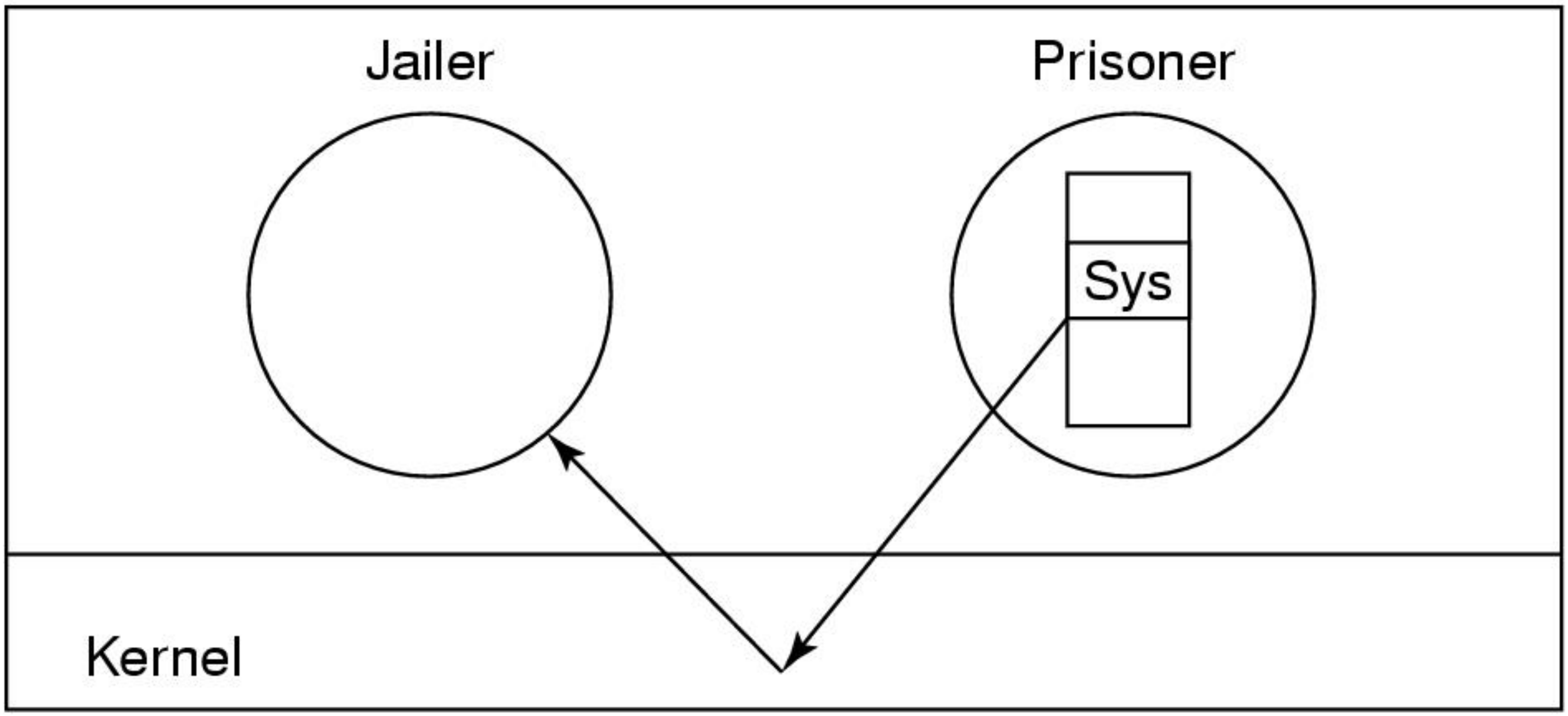


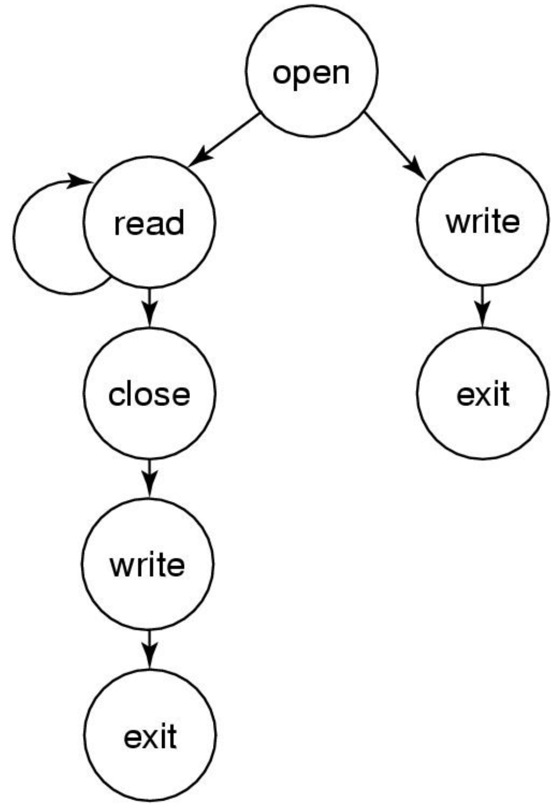
Figure 9-35. The operation of a jail.

Model-Based Intrusion Detection

```
int main(int argc *char argv[])
{
  int fd, n = 0;
  char buf[1];

  fd = open("data", 0);
  if (fd < 0) {
      printf("Bad data file\n");
      exit(1);
  } else {
      while (1) {
           read(fd, buf, 1);
           if (buf[0] == 0) {
             close(fd);
             printf("n = %d\n", n);
             exit(0);
           }
        n = n + 1;
      }
  }
}
```

(a)



(b)

Figure 9-36. (a) A program. (b) System call graph for (a).

Sandboxing

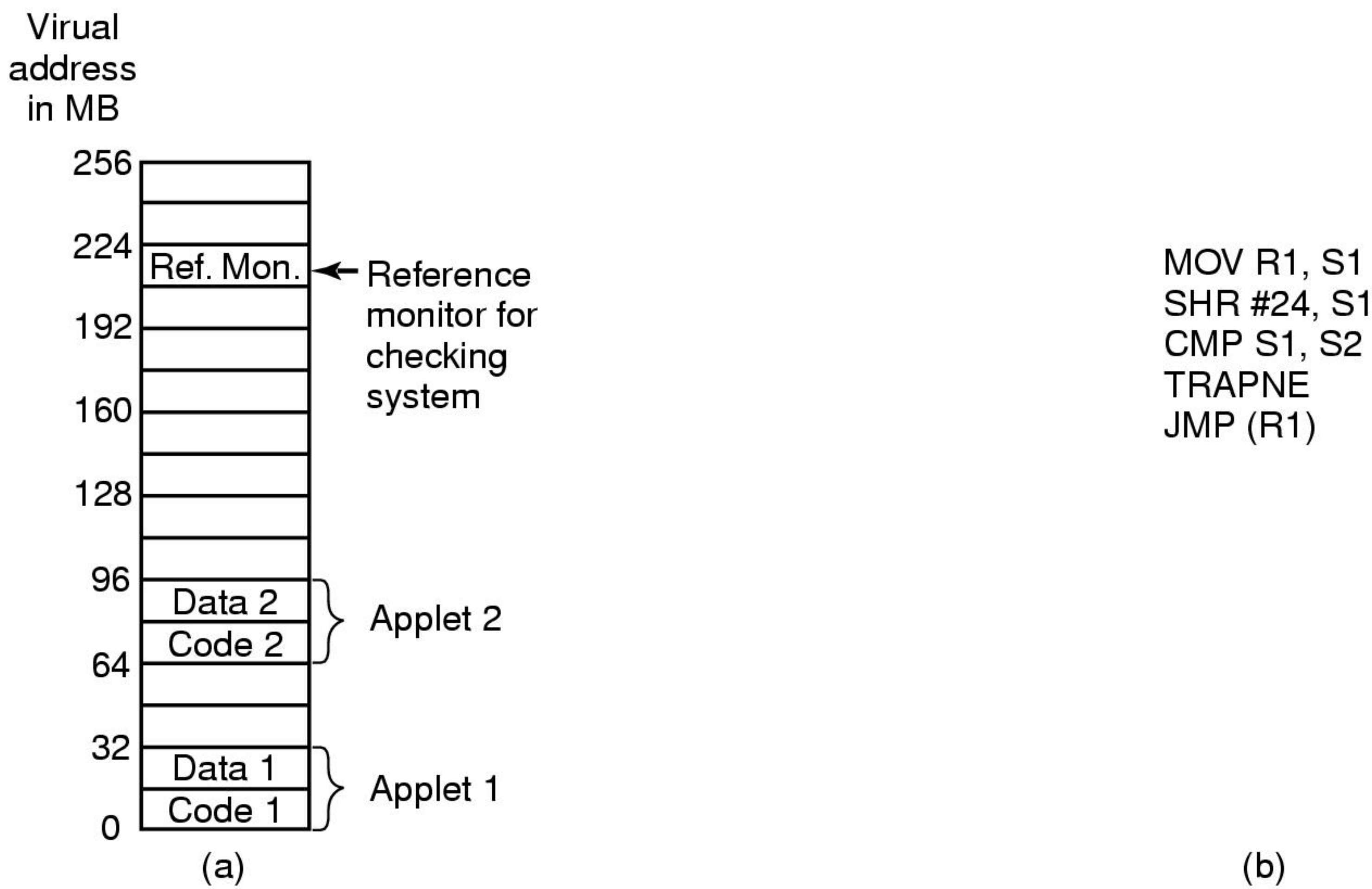


Figure 9-37. (a) Memory divided into 16-MB sandboxes. (b) One way of checking an instruction for validity.

Interpretation

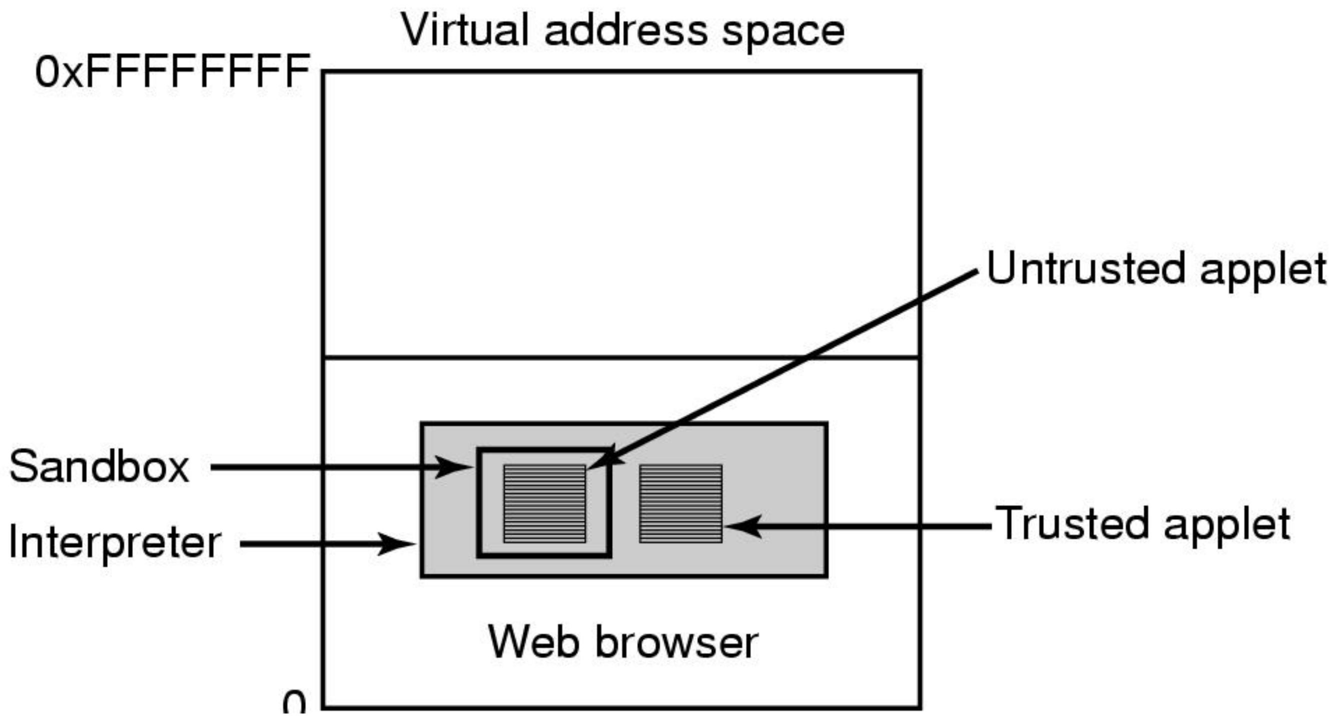


Figure 9-38. Applets can be interpreted by a Web browser.

Java Security (1)

JVM byte code verifier checks if the applet obeys certain rules:

- Does the applet attempt to forge pointers?
- Does it violate access restrictions on private-class members?
- Does it try to use a variable of one type as another type?
- Does it generate stack overflows? underflows?
- Does it illegally convert variables of one type to another?

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

Figure 9-39. Some examples of protection that can be specified with JDK 1.2.