Chapter 9
Security
## Threats

<table>
<thead>
<tr>
<th>Goal</th>
<th>Threat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data confidentiality</td>
<td>Exposure of data</td>
</tr>
<tr>
<td>Data integrity</td>
<td>Tampering with data</td>
</tr>
<tr>
<td>System availability</td>
<td>Denial of service</td>
</tr>
<tr>
<td>Exclusion of outsiders</td>
<td>System takeover by viruses</td>
</tr>
</tbody>
</table>

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:

- 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.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
Secret-Key Cryptography

Monoalphabetic substitution:

Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ciphertext: QWERTYUIOPasdfghjklzxcvbnm
Public-Key Cryptography

• Encryption makes use of an "easy" operation, such as how much is $314159265358979 \times 314159265358979$?

• 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)

**Figure 9-5. A protection matrix.**

<table>
<thead>
<tr>
<th>Domain</th>
<th>File1</th>
<th>File2</th>
<th>File3</th>
<th>File4</th>
<th>File5</th>
<th>File6</th>
<th>Printer1</th>
<th>Plotter2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>Read</td>
<td>Write</td>
<td>Read</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Read</td>
<td></td>
<td></td>
<td>Read</td>
<td>Write</td>
<td>Write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Read</td>
<td>Write</td>
<td>Execute</td>
<td></td>
<td>Write</td>
<td></td>
<td>Write</td>
</tr>
</tbody>
</table>
## Protection Domains (3)

![Protection Matrix]

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

<table>
<thead>
<tr>
<th>Domain</th>
<th>File1</th>
<th>File2</th>
<th>File3</th>
<th>File4</th>
<th>File5</th>
<th>File6</th>
<th>Printer1</th>
<th>Plotter2</th>
<th>Domain1</th>
<th>Domain2</th>
<th>Domain3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>Read</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td></td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>Read</td>
<td></td>
<td>Read</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 9-7. Use of access control lists to manage file access.
### Access Control Lists (2)

![Image of a page from a book](Image)

#### Figure 9-8. Two access control lists.

<table>
<thead>
<tr>
<th>File</th>
<th>Access control list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>tana, sysadm: RW</td>
</tr>
<tr>
<td>Pigeon_data</td>
<td>bill, pigfan: RW; tana, pigfan: RW; ...</td>
</tr>
</tbody>
</table>
Figure 9-9. When capabilities are used, each process has a capability list.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
Capabilities (2)

<table>
<thead>
<tr>
<th>Server</th>
<th>Object</th>
<th>Rights</th>
<th>f(Objects,Rights,Check)</th>
</tr>
</thead>
</table>

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.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
## Formal Models of Secure Systems

*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.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
Covert Channels (2)

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

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 elksi
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

<table>
<thead>
<tr>
<th>User</th>
<th>Password</th>
<th>Salt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bobbie</td>
<td>4238</td>
<td>e(Dog, 4238)</td>
</tr>
<tr>
<td>Tony</td>
<td>2918</td>
<td>e(6%%TaeFF, 2918)</td>
</tr>
<tr>
<td>Laura</td>
<td>6902</td>
<td>e(Shakespeare, 6902)</td>
</tr>
<tr>
<td>Mark</td>
<td>1694</td>
<td>e(XaB#Bwcz, 1694)</td>
</tr>
<tr>
<td>Deborah</td>
<td>1092</td>
<td>e(LordByron, 1092)</td>
</tr>
</tbody>
</table>

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

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
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.
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);

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, "zzzz") == 0) break;
}
execute_shell(name);

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

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.
Figure 9-25. (a) The stack before the attack. (b) The stack after the stack has been overwritten.
int main(int argc, char *argv[]) {
    char src[100], dst[100], cmd[205] = "cp ";
    printf("Please enter name of source file: ");
    gets(src);
    strcat(cmd, src);
    strcat(cmd, " ");
    printf("Please enter name of destination file: ");
    gets(dst);
    strcat(cmd, dst);
    system(cmd);
}
/* declare 3 strings */
/* ask for source file */
/* get input from the keyboard */
/* concatenate src after cp */
/* add a space to the end of cmd */
/* ask for output file name */
/* get input from the keyboard */
/* complete the commands string */
/* 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
Figure 9-27. A recursive procedure that finds executable files on a UNIX system.
Figure 9-27. A recursive procedure that finds executable files on a UNIX system.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
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.
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.
Figure 9-31. A simplified view of a hardware firewall protecting a LAN with three computers.
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.
### Figure 9-33. Examples of a polymorphic virus.

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,R1</td>
<td>MOV A,R1</td>
<td>MOV A,R1</td>
<td>MOV A,R1</td>
<td>MOV A,R1</td>
</tr>
<tr>
<td>ADD B,R1</td>
<td>NOP</td>
<td>ADD #0,R1</td>
<td>OR R1,R1</td>
<td>TST R1</td>
</tr>
<tr>
<td>ADD C,R1</td>
<td>ADD B,R1</td>
<td>ADD B,R1</td>
<td>ADD B,R1</td>
<td>ADD C,R1</td>
</tr>
<tr>
<td>SUB #4,R1</td>
<td>NOP</td>
<td>OR R1,R1</td>
<td>MOV R1,R5</td>
<td>MOV R1,R5</td>
</tr>
<tr>
<td>MOV R1,X</td>
<td>ADD C,R1</td>
<td>ADD C,R1</td>
<td>ADD C,R1</td>
<td>ADD B,R1</td>
</tr>
<tr>
<td>NOP</td>
<td>SHL #0,R1</td>
<td>SHL R1,0</td>
<td>CMP R2,R5</td>
<td></td>
</tr>
<tr>
<td>SUB #4,R1</td>
<td>SUB #4,R1</td>
<td>SUB #4,R1</td>
<td>SUB #4,R1</td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>JMP .+1</td>
<td>ADD R5,R5</td>
<td>ADD R5,R5</td>
<td></td>
</tr>
<tr>
<td>MOV R1,X</td>
<td>MOV R1,X</td>
<td>MOV R1,X</td>
<td>MOV R1,X</td>
<td>MOV R1,X</td>
</tr>
<tr>
<td>MOV R5,Y</td>
<td>MOV R5,Y</td>
<td>MOV R5,Y</td>
<td>MOV R5,Y</td>
<td></td>
</tr>
</tbody>
</table>

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
Antivirus and Anti-Antivirus Techniques

- Virus scanners
- Integrity checkers
- Behavioral checkers
- Virus avoidance
Figure 9-34. How code signing works.

- **Software vendor**
  - Program
  - Signature
  - Signature generation
    - \( H = \text{hash}(\text{Program}) \)
    - Signature = encrypt(\( H \))

- **User**
  - Program
  - Signature
  - Signature verification
    - \( H_1 = \text{hash}(\text{Program}) \)
    - \( H_2 = \text{decrypt}(\text{Signature}) \)
    - Accept Program if \( H_1 = H_2 \)
Jailing

Figure 9-35. The operation of a jail.
Figure 9-36. (a) A program. (b) System call graph for (a).
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)

<table>
<thead>
<tr>
<th>URL</th>
<th>Signer</th>
<th>Object</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.taxprep.com">www.taxprep.com</a></td>
<td>TaxPrep</td>
<td>/usr/susan/1040.xls</td>
<td>Read</td>
</tr>
<tr>
<td>*</td>
<td>TaxPrep</td>
<td>/usr/tmp/*</td>
<td>Read, Write</td>
</tr>
<tr>
<td><a href="http://www.microsoft.com">www.microsoft.com</a></td>
<td>Microsoft</td>
<td>/usr/susan/Office/*</td>
<td>Read, Write, Delete</td>
</tr>
</tbody>
</table>

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