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**

<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>
</tbody>
</table>

**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

(a) A successful login
(b) Login rejected after name entered
(c) Login rejected after name and password typed
Authentication Using Passwords

```bash
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

### The use of salt to defeat precomputation of encrypted passwords

<table>
<thead>
<tr>
<th>Name</th>
<th>Password</th>
<th>Encrypted Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bobbie</td>
<td>4238</td>
<td>e(Dog4238)</td>
</tr>
<tr>
<td>Tony</td>
<td>2918</td>
<td>e(6%%TaeFF2918)</td>
</tr>
<tr>
<td>Laura</td>
<td>6902</td>
<td>e(Shakespeare6902)</td>
</tr>
<tr>
<td>Mark</td>
<td>1694</td>
<td>e(XaB@Bwcz1694)</td>
</tr>
<tr>
<td>Deborah</td>
<td>1092</td>
<td>e(LordByron,1092)</td>
</tr>
</tbody>
</table>
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.

CuuDuongThanCong.com
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
  - if 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);

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);

(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 no 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
HowVirusesWork(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”
Recursive procedure that finds executable files on a UNIX system

Virus could infect them all

#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>

struct stat sbuf;

search(char *dir_name)
{
    DIR *dirp;
    struct dirent *dp;
    dirp = opendir(dir_name);
    if (dirp == NULL) return;
    while (TRUE) {
        dp = readdir(dirp);
        if (dp == NULL) {
            chdir ("..");
            break;
        }
        if (dp->d_name[0] == '.') continue; /* skip the . and .. directories */
        lstat(dp->d_name, &sbuf);
        if (S_ISLNK(sbuf.st_mode)) continue; /* skip symbolic links */
        if (chdir(dp->d_name) == 0) {
            search(".");
        } else {
            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

### Examples of a polymorphic virus

All of these examples do the same thing.

<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 B,R1</td>
<td>MOV R1,R5</td>
</tr>
<tr>
<td></td>
<td>NOP</td>
<td>SHL #0,R1</td>
<td>SHL R1,0</td>
<td>ADD C,R1</td>
</tr>
<tr>
<td></td>
<td>ADD #4,R1</td>
<td>SUB #4,R1</td>
<td>SUB #4,R1</td>
<td>ADD B,R1</td>
</tr>
<tr>
<td></td>
<td>MOV R1,X</td>
<td>JMP .+1</td>
<td>JMP .+1</td>
<td>CMP R2,R5</td>
</tr>
<tr>
<td></td>
<td>MOV R1,X</td>
<td>MOV R5,R5</td>
<td>MOV R1,X</td>
<td>MOV R5,R5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MOV R5,Y</td>
<td>MOV R5,Y</td>
</tr>
</tbody>
</table>

CuDuongThanCong.com

https://fb.com/tailieudientucntt
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
Applets can be interpreted by a Web browser
Mobile Code (3)

How code signing works

Software vendor

Applet

Signature

Signature generation

H = hash(Applet)
Signature = encrypt(H)

User

Applet

Signature

Signature verification

H1 = hash(Applet)
H2 = decrypt(Signature)
Accept Applet if H1 = H2

Internet
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)

Examples of specified protection with JDK 1.2

<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></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>
Protection Mechanisms

Protection Domains (1)

Examples of three protection domains
### Protection Domains (2)

<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 Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Read</td>
<td>Read Write Execute</td>
<td>Read Write</td>
<td></td>
<td></td>
<td>Write</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>Read Write Execute</td>
<td>Write</td>
<td></td>
<td>Write</td>
<td>Write</td>
</tr>
</tbody>
</table>

### A protection matrix

- Domain 1: Read, Read Write
- Domain 2: Read, Read Write Execute, Read Write, Write
- Domain 3: Read Write Execute, Write, Write
Protection Domains (3)

A protection matrix with domains as objects

<table>
<thead>
<tr>
<th></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>Read</td>
<td>Read</td>
<td>Read</td>
<td>Read</td>
<td>Enter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Write</td>
<td>Read</td>
<td>Write</td>
<td>Write</td>
<td>Write</td>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CuuDuongThanCong.com

https://fb.com/tailieudientucntt
Access Control Lists (1)

Use of access control lists of manage file access
### Access Control Lists (2)

<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>

**Two access control lists**
Capabilities (1)

Each process has a capability list
Capabilities (2)

- Cryptographically-protected capability

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

- Generic Rights
  1. Copy capability
  2. Copy object
  3. Remove capability
  4. Destroy object
Trusted Systems
Trusted Computing Base

A reference monitor

User process

All system calls go through the reference monitor for security checking
**Formal Models of Secure Systems**

<table>
<thead>
<tr>
<th></th>
<th>Compiler</th>
<th>Mailbox 7</th>
<th>Secret</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Eric</strong>&lt;br&gt;Read Execute</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Henry</strong>&lt;br&gt;Read Execute</td>
<td>Read Write</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Robert</strong>&lt;br&gt;Read Execute</td>
<td>Read Write</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) An authorized state

(b) An unauthorized state

![Diagram](https://fb.com/tailieudientucntt)
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)

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

- Symbol X means new requirements
- Symbol -> requirements from next lower category apply here also
# Orange Book Security (2)

<table>
<thead>
<tr>
<th>Assurance</th>
<th>System architecture</th>
<th>System integrity</th>
<th>Security testing</th>
<th>Design specification and verification</th>
<th>Covert channel analysis</th>
<th>Trusted facility management</th>
<th>Configuration management</th>
<th>Trusted recovery</th>
<th>Trusted distribution</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Documentation</th>
<th>Security features user’s guide</th>
<th>Trusted facility manual</th>
<th>Test documentation</th>
<th>Design documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>X, X, X</td>
<td>X, X, X</td>
<td>X, X, X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

CuuduongThanCong.com  
https://fb.com/tailieudientucntt
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