Security
Security in the “Real” World

- We are all familiar with securing valuables
  - Guards
  - Locked doors, cabinets, safes
  - ID badges
- Goal: Only authorised people have access to the valuables
- How does this relate to computer systems?
Computer System “Valuables”

• Hardware
  – Threats include theft, accidental or deliberate damage.
  – Hardware security is similar to *physical security* of valuables
    • Use similar techniques to secure the physical hardware.
Computer System “Valuables”

• Data
  – Three general goals of data security
  – Confidentiality
    • Data is only readable by authorised people
      – Able to specify who can read what on system, and be enforced
      – Preserve secrecy or privacy
  – Integrity
    • Data is only modifiable by authorised people
  – Availability
    • Data is available to authorized parties
Overloading a server or network
Disabling a file server
Cutting a communication line
 Destruction of hardware

Example:

Attack on Availability
becomes unavailable or unusable
An asset of the system is destroyed, or

Denial of Service

Threats
Threats

• Interception
  – An unauthorised party gains access to an asset
  – Attack on Confidentiality
  – Examples:
    • Wiretapping to capture data on a network
    • Illicit copying of files and programs
Threats

• Modification
  – An unauthorized party not only gained access, but tampers with data
  – Attack on Integrity
  – Examples:
    • Changing values in a file
    • Altering a program so that it performs differently
    • Modifying the content of messages being transmitted on a network
Data Security

• Can be partially solved using physical security
• Usually too expensive or inconvenient to do so
  – Example:
    • Each user has private computer, in a locked guarded room.
    • No sharing of information is permitted
    • No outside connectivity permitted
      – No email, shared file server, shared printer, shared tape drive
      – No printouts or storage media can enter or exit the room.
    • Users can still memorise information a bit at a time and leak secrets
• However, physical security is still an important part of any computer security system.
Intruders

- Strategies to provide security typically consider the expected *intruders* (also called *adversaries*) to be protected against.

- Common categories
  1. Casual prying by non-technical users
     - Stumble across others users’ files on file server
  2. Snooping by insiders
     - Local programmer explicitly attempting to break security
  3. Determined attempts to make money
     - Bank programmers installing software to steal money
  4. Commercial or military espionage
     - Well funded attempts to obtain corporate or government secrets

- Depending on the value of the data, and the perceived adversary,
  - more resources may be provided to secure the system
  - less convenient methods of access may be tolerated by users
Data Loss

- Protecting against data loss is an important part of any security policy
- Examples:
  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
- General approach is off-site backups
User Authentication

• Thus far, we have described various concepts with reference to authorised users
• Assume we can decide whether a given user is authorised to perform an operation, but how can we determine if the user is who he says he is?
  ⇒ How can we authenticate the users?
Approaches to User Authentication

- Three general approaches to identifying a user
  - Based on some unique property they possess
    1. Something the user knows
    2. Something the user has
    3. Something the user is
  - Each approach has its own complexities and security properties
Authentication Using Passwords

- Most common form of authentication is entering a login name and password
  - The password entered is not displayed for obvious reasons
  - Windows 2K/XP is broken in this regard
    - Prints ‘*’ for each character typed
      - Reveals the length of password
    - Also remembers the last login name
  - UNIX approach is much better
    - In security, the less revealed the better
Example: Less is More

- Careless login program can give away important information
  a) Successful login
  b) Valid login ID revealed
  c) No useful information revealed

<table>
<thead>
<tr>
<th>LOGIN: ken</th>
<th>LOGIN: carol</th>
<th>LOGIN: carol</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD: FooBar</td>
<td>INVALID LOGIN NAME</td>
<td>PASSWORD: Idunno</td>
</tr>
<tr>
<td>SUCCESSFUL LOGIN</td>
<td>LOGIN:</td>
<td>INVALID LOGIN</td>
</tr>
</tbody>
</table>

(a)                     (b)                     (c)

COMP3231
Problems with Password Security

• One study from 1979
  – Given a list of first name, last names, street names, moderate dictionary, licence plate number, some random strings, the previous spelt backwards, etc..
  – A comparison with a password file obtained 86% of all passwords

• A more recent study (1990) produced similar results
The Importance Password Security

• Good password security is vital if computer is publicly accessible.
  – E.g., dialup server
  – Connected to a network or the Internet

• It’s common for war diallers to probe phone numbers or crackers to probe internet connect machines
Approaches to improving password security

- Passwords are stored encrypted
  - Avoids sysadmins, and potentially unwanted computer “maintainers” from obtaining passwords
    - Example: from backup tapes
- Login procedure takes user-supplied string,
  - encrypts it
  - compares result to stored encrypted password
An Attack on Encrypted Passwords

• Take the dictionary of words, names, etc, and encrypt all of them using the same encryption algorithm

• Simply match pre-encrypted list with password file to get matches
Improving Password Security with a Salt

- **Idea:**
  - Encrypt the password together with a n-bit random number (the *salt*) and store both the number and encrypted result
  - Example
    \[
    \text{result} = e('Dog1234'), 1234
    \]

- **Cracker must encrypt each dictionary word }2^n\text{ different ways**
  - Make pre-computed list }2^n\text{ times larger

- **UNIX takes this approach with }n = 12\text{**

- **Additional security via making encrypted passwords unreadable (shadow passwords)**
Improving Password Security

• Storing passwords more securely does not help if user ‘homer’ has the password ‘homer’

• Users must be educated (or forced) to choose good passwords
  – Approaches:
    • Warn users who choose poor passwords
    • Pick passwords for users
      – easy to remember nonsense words
    • Force them to change the password regularly
Issues with ‘Good’ Passwords

- By forcing frequent password changes, users tend to choose simpler passwords.
- By choosing too ‘good’ a password for users, users put them on post-it notes on the monitor.
- Still many attacks involving intercepting password between user and service.
Aside: One-Way Functions

• Function such that given formula for $f(x)$
  – easy to evaluate $y = f(x)$
• But given $y$
  – computationally infeasible to find $x$
One-time Passwords

- Password changing in the extreme
- Advantage:
  - Snooping login provides no useful information
    - Only a stale previous password
- Approach:
  - Choose a secret phrase and the number of one time passwords required.
  - Each password is generated via re-applying a one-way function
  - Passwords are then used in reverse order
    - Easy to compute the previous password, but not the next.
One-time Password: Example

- $P_0 = f(f(f(f(s))))$
- $P_1 = f(f(f(s)))$
- $P_2 = f(f(s))$
- $P_3 = f(s)$

- Server initially stores $P_0$
- Server receives O-T password ($P$) and computes $f(P)$
- If $f(P)$ matches $P_0$, login successful, server stores $P (= P_1)$

- On home PC
  - Compute one-time password to supply via 3 iterations of 1 way function
  - Subsequent via 2, 1, 0

- Note
  - Server never stores secret ($s$)
  - Home PC store number of passwords used, but does not need to store secret either.

THE UNIVERSITY OF NEW SOUTH WALES

COMP3231 24
Challenge-Response

- Server and client both know secret key \( (k) \)
- Server sends a *challenge* -- random number \( (c) \) to client
- Client combines the secret key \( (k) \) with random number \( (c) \) and applies a publicly-known function \( r = f(c,k) \)
- Client sends the response to server
- On server, if supplied \( r \) equals \( f(c,k) \) we have successful login
Challenge-Response

- **Advantage:**
  - Secret Key is never transmitted on potentially insecure networks
  - Eavesdropping is fruitless
    - Assuming function \( f \) is such that \( k \) cannot be easily deduced from a large number of observed challenge-responses

- **Con:**
  - Need a ‘computer’ present to login (compute response)
    - PDA, phone, etc.
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.
- Alternatives:
  - Retina scans
  - Voice analysis
  - Analysing signature dynamics
Issue: User Acceptance

- Low user acceptance results in:
  - Users themselves compromising the system
    - Example: using post-it notes
  - Refusal to login
    - E.g., login using a blood sample

- Challenge:
  - To find a secure, unobtrusive, simple scheme
Authentication Summary

• Authentication is an important component of security
• Password-based schemes only modestly robust to attack. Many attacks possible
  – Insecure user behaviour
  – Password storage
  – Attacks on cryptographic algorithms (for storage or transfer)
    – Snooping Networks
• Physical and Biometric authentication improves security
  – Attacks still possible, but more resources required.
Software Threats

• Given an reasonable authentication mechanism, many other software threats exist.
• Software Exploits
  – Trojan Horses
  – Login Spoofing
  – Logic Bombs
  – Trapdoors
  – Buffer Overflows
• Self replicating
  – Viruses
Trojan Horses

- Seemingly innocent program executed by an unsuspecting user
  - Either directly or indirectly
- Program can then do anything the user can
  - Modify or delete files, send them elsewhere on the net.
- Sample exploit
  - If a user has ".", "/bin" or similar in their PATH, place a file called ls in your directory (or /tmp).
Login Spoofing

- Write a program that emulates the login screen.
- Login, run the program to collect password of unsuspecting user, then exit to the real login prompt.
- Windows 2K/XP provides a key combination (CTRL-ALT-DEL) that can’t be bypassed to produce the real login program (Secure Attention Key).
Logic Bombs

- Code secretly embedded in an application or the OS that *goes off* when certain conditions are met.
  - Example: Payroll programmer embeds code that checks he is on the payroll, if not, the payroll software becomes malicious
Trap Doors

- Code inserted by the programmer to bypass some check.
  - Example: The login program

```c
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);
```

```c
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);
```
Buffer Overflows

- *Main* calls *A* which has a local buffer
- Overflow the buffer with code + starting address of the code
- Good for both local and remote attacks
- Caused by programmers not checking buffer bounds
Viruses

- A program that reproduces itself by attaching its code to another program.
- Can do anything the normal program could do
  - Print harmless message
  - Destroy all files on hard disk
  - Send all your data to the net
  - Trash the EEPROM BIOS to make your computer inoperable
  - Denial of service attack
How Viruses Work

• Virus written in assembly language
• Inserted into another program
• Virus dormant until program executed
  – then infects other programs
  – eventually executes its “payload”
How Viruses Work

- **Parasitic Viruses**
  - Add their code to various locations in the executable
  - Redirect the start address in the header
  - On execution, it may replicate by modifying another executable file (and other malicious activities).
How Viruses Work

• Boot Sector Viruses
  – Copies original boot block to different location
  – Replaces boot block with itself
  – When machine boots, virus is loaded into RAM
  – It installs itself, and then boots OS via original boot block

• How does it regain control later?
How Viruses Work

- Virus installs interrupt handlers which rely on OS not installing all its own handlers prior to next interrupt occurring
  - Older versions of Windows behaved that way
- Virus re-installs trap handlers at next opportunity
How Viruses Work

- Memory Resident Viruses
  - Install themselves in main memory
  - Typically redirect the exception/interrupt handlers to itself
    - Still calls the real code to remain undetected
    - checks and reinstall redrections changed
    - Replicate during, or manipulate and spy-on on syscalls

![Diagram](image.png)
How Viruses Work

• Macro Viruses
  – Rely on overly powerful/feature overloaded macro languages
  – MS office uses visual basic – complete programming language that can read/write files
  – Opening a Word document is like running a program (it could do anything)
How Viruses Spread

• Virus placed where it’s 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 (address book) to replicate
Antivirus Approach

• Scanning
  – Search each file and check if virus present
    • 10,000 potential viruses and 10,000 files
    • Hard to make fast
  – Use fuzzy searches to catch small changes in known viruses
    • Slower, false positives
  – Trade-off between accuracy and acceptable performance
Antivirus and Anti-Antivirus Techniques

(a) A program
(b) Infected program
Change in file length a give away
Antivirus and Anti-Antivirus Techniques

(c) Compressed infected program
Presence of virus code still a give away
Antivirus and Anti-Antivirus Techniques

(d) Encrypted virus
Presence of (de)compressor a give away
Antivirus and Anti-Antivirus Techniques

<table>
<thead>
<tr>
<th>(a) Executable program</th>
<th>(b) Executable program</th>
<th>(c) Original size</th>
<th>(d) Original size</th>
<th>(e) Original size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Header</td>
<td>Unused</td>
<td>Virus</td>
<td>Virus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compressed</td>
<td>Compressor</td>
<td>Encrypted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>executable program</td>
<td></td>
<td>Virus</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Compressor</td>
<td>Decryptor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Decryptor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Compressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>executable program</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Header</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(e) Compressed virus with encrypted compression code
Can still search for remaining decryptor code
Antivirus and Anti-Antivirus Techniques

Examples of a polymorphic virus

All of these examples do the same thing
Antivirus and Anti-Antivirus Techniques

• Integrity checkers
  – Scan the disk and determine checksums for all executable files
  – Check checksums, if changed we have a virus
  – Counter, viruses can hack checksum database is

• Behavioural checkers
  – Look for virus-like behaviour
    • Example: overwriting executable file
      – False alarms (e.g., a compiler)
Antivirus and Anti-Antivirus Techniques

• Virus avoidance
  – good OS
    • Separate user/system mode/protection to minimise damage
  – Run/install only reputable software
  – use antivirus software
  – Do not open attachments to email
  – frequent backups

• Recovery from virus attack
  – halt computer, reboot from safe disk, run antivirus
  – restore from backups
Running Foreign Code

- We can see that running foreign code can be dangerous (Trojan horse, viruses, simply malicious, etc.)
- Problem is that all the code we run has all the privileges we do
- We need a method of running untrusted code safely
Principle of Least Privilege

• A guiding principle we would like to apply
• Idea:
  – Give the suspicious program only the privileges required to complete the task you expect, nothing more
  – Example:
    • Can only perform file related system calls
    • Can only access files within a specified directory
Example: Active Web Content

- We’d like to browse “active” web content
  - Run content in the web browser
  - The browser has all the privileges we do

- Some approaches
  - Sandboxing
  - Interpretation
  - Code Signing
Sandboxing

**Idea:**
- Code runs within a sandbox.
- Code runs within a browser (or some other larger application).
- The applet can access only the data contained within its sandbox, and nothing else.
- It can only jump to code within its sandbox (and cannot modify the code).

**How can we create a sandbox within a process?**
Sandbox Implementation

• Firstly, assume we can restrict access to code to avoid problem of self modifying code

• To restrict code to the code segment
  – Scan the code
  – Check all jumps and branches jump to addresses within the sandbox
    • Handle both absolute and relative addresses
  – For computed (dynamic jumps) we insert extra instruction into the code to check the destination addresses are within the code
    • Involves fairly complex code rewriting, but it is doable

• To restrict data access to data section, we do the same thing we did for code
Sandbox Implementation

• What about system calls
  – We use a reference monitor that
    • Intercepts all system calls
    • Determine whether the call is allowed to succeed or not
      – Based on the type of call, or the arguments supplied.
  – Reference monitor restricts the system calls to a safe subset
Interpretation

- Instead of running code directly (natively), we run it using an interpreter
  - Interpreter can apply addressing restrictions
  - Can consider the interpreter as implementing a sandbox
  - Example: JAVA
Code Signing

- Authenticity of the code is guaranteed
- Issues
  - Does not protect you against bad or buggy code
  - Example: Shockwave has had various “authentic” security problems
Summary

• Even given strong authentication, there are many software threats to data security policies.
• The affect of exploiting those threats can be minimised by adopting the principle of least privilege.