Protection (محافظت) and Security (امنیت)

Operating System Course
Department of Electrical and Computer Engineering,
Shiraz University

Spring 2010
Course Instructor: Reza Azimi
Protection System

- **Who** can access **What**
- **How** they can access it
- **Objects** are “what”, **Subjects** are “who”, **Actions** are “how”

A protection system dictates whether a given **action** performed by a given **subject** on a given **object** should be allowed:
- you can read and/or write your files, but others cannot
- you can read “/etc/passwd”, but you cannot write it
Types of Access

- None
- Knowledge (e.g. file existence)
- Execute
- Read
- Append
- Update
- Change Protection
- Create/Delete
Representing Protection

**Access Control Lists (ACL)**
For each object, maintain a list of subjects and their permitted actions

<table>
<thead>
<tr>
<th>Subjects</th>
<th>/usr</th>
<th>/sbin</th>
<th>/tmp</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysadm</td>
<td>rwx</td>
<td>rwx</td>
<td>rwx</td>
</tr>
<tr>
<td>joeuser</td>
<td>rx</td>
<td>x</td>
<td>rwx</td>
</tr>
<tr>
<td>guest</td>
<td>x</td>
<td>-</td>
<td>rx</td>
</tr>
</tbody>
</table>

**Capabilities**
For each subject, maintain a list of objects and their permitted actions
ACL’s and Capabilities

- Differ only in how the subject-object-capability table is represented
  - which approach does Unix use?

- **Capabilities**
  - easier to transfer
    - they are like keys, can handoff, do not depend on subject
  - revocation of capabilities requires keeping track of all subjects that have the capability – a challenging problem

- **ACLs**
  - easier to manage
  - ACL’s object-centric, easy to grant and revoke
  - have a problem when objects are heavily shared
    - ACLs grow very large
    - mitigate using groups (e.g., Unix)
Access Rights in Unix

- File owner can specify three access rights:
  - read
  - write
  - execute

- Appears as `rwxrwxrwx` (when you “ls -l”) for:
  - **Self** (user) – e.g. prevent accidental deletion
  - a **Group** in /etc/group (**group**) – e.g. CVS repo
  - **All** other users (**other**) – e.g. public .html files
Protection in Unix

- User Id:
  - Unique for each subject (process)
  - Each object (file, socket, etc) is owned by a unique user

- Group Id:
  - Several ids for subjects (processes)
  - Each object (file, socket, etc) belongs to a unique group

- Object ACLs:
  - Specify read, write, and execute permissions for user, group, and others.
  - A subject can r,w,x an object if:
    - it has the same uid and the appropriate user permission is set
    - one of its gid’s is the same as that of the object and the appropriate group permission is set; or
    - the appropriate other permission is set.
  - Super-users (uid root) can do anything they want
OS Role in Security

- Prevent malicious software from
  - Accessing/changing users’ data
  - Crashing the system, or making the system unusable
  - Taking over the system, exploiting it
    - For distributed denial of service attack
    - “spying”

- Malware
  - Viruses
  - Trojan Horses
  - Worms
  - Spyware
Security Goals

- Confidentiality
  - Preventing the unauthorized release of information

- Integrity
  - Preventing unauthorized modification of information

- Authenticity
  - Messages come from who they purport to be from

- Availability
  - Deflect/w withstand denial-of-service attacks
Trusted Computing Base (TCB)

Q: What are you trusting with your information?

Example:
- If you type your password on a keyboard, you’re trusting:
  - The keyboard manufacturer
  - Your computer manufacturer
  - Your operating system
  - The password library
  - The application that’s checking the password
  - The fact that none of these have been tampered with
Trusted Computing Base (TCB)

- Set of components (hardware and software) that you trust your secrets with

- What should be in your TCB?
  - Public Web kiosks?
  - Your operating system?
  - How about your compiler?
Buffer Overflows

- One of the most commonly used mechanisms for attacks.
- Suppose a web server contains a function:

```c
void func(char *str) {
    char buf[128];
    strcpy(buf, str);
    do-something(buf);
}
```

- When the function is invoked the stack looks like:

- What if *str is 136 bytes long? After `strcpy`:

---

*Operating System Concepts, Shiraz University, Spring 2010, by Reza Azimi*
Stack Smashing Exploit

- Main problem: no range checking in `strcpy()`.
- Attacker puts code in buffer such that after `strcpy` stack looks like:

  ![Stack Diagram]

  Program P: `exec( "/bin/sh" )`

- When `func()` exits, the user will be given a shell !!
  - Note: attack code runs in the stack.
  - To determine `ret` the attacker must correctly guess position of stack when `func()` is called.
Boot Sector Virus

Boot Sector Virus

Boot Sector Other Sectors

Bootstrap Loader System Init System Init Bootstrap Loader

chain

before infection

Boot Sector Other Sectors

Virus Code System Init System Init Bootstrap Loader

chain chain

after infection

14 Operating System Concepts, Shiraz University, Spring 2010, by Reza Azimi
Security Design Principles

- Security is much more than just cryptography!
  - if there are flaws in the system's design, all the cryptography in the world can't rescue you

- Principle of “Least Privilege”
  - Figure out which capabilities a program or service needs, and only grant it those!
  - Not always easy:
    - start by running program, see if it works, if not, add more privileges, iterate ...
    - obviously can't be sure you have caught all of the requirements this way
Principle of Least Common Mechanism

- Basic lesson: **Be careful with shared code**
  - Assumptions on its uses may not be valid in different contexts

- Example: Internet Explorer HTML Rendering code
  - IE exports an API to render HTML in a window
  - used by other applications to display HTML content, e.g., Outlook
Principle of Least Common Mechanism

- HTML rendering code knows that Java applets loaded from the Net can be unsafe
  - By default wisely disables Java execution in HTML content when loading from Internet

- However, Java is enabled for HTML pages loaded from disk!
  - Outlook fetch the email to render from disk ..
Principle of Least Authority

- It is considered good security practice to allow a subject access to information or authority to do anything, only when it is necessary for his doing the job in question.
- Ideally this would be enforced by a security policy and by the operating system supporting it.
Principle of Complete Mediation

- Idea: Check *every access* to every object
  - in rare cases, can get away with less (e.g., caching)
  - but must be sure that nothing relevant in the environment has changed!

- Example:
  - Web server encodes session state in URL or form sent back to client
  - Client can easily see and change their state on the server!
  - example: Shopping cart with URL-exposed price total
Security through Obscurity

- **Idea:**
  - Don’t disclose algorithms or implementation details of the system, making it harder for someone to find vulnerabilities

- **Counterargument:**
  - A system becomes more secure as more people scour over its details and find flaws!
    - **Claim:** A system should be secure even if all of its implementation details are public
Encryption

- Symmetric (or shared-key) algorithms:
  - Both “Alice” and “Bob” know the encryption/decryption key
  - They shared the key earlier through some safe communication channel

- Public-key algorithms:
  - Mathematically interlinked encrypt/decrypt keys
  - Decryption key cannot be calculated from the encryption key
  - Therefore the encryption key can be public!
    - *To send Bob a confidential message, encrypt it with Bob's public key, which is on his website...*
Symmetric Key Encryption

- Data Encryption Standard (DES)
  - Standard algorithm, widely used since late '70s
  - Operates on 64-bit blocks of data & 56-bit key
  - Lots of simple operations applied on a block many times

- Very efficient to compute
- Implemented in hardware
  - Encryption at full network speed (e.g. 1 Gbps).
Symmetric Key Problem

how do you securely distribute the shared keys?

- Key Distribution Centre
- Use public/private encryption to distribute symmetric keys securely.
Public Key Cryptography

- **Idea:**
  - Widely advertise the encryption key, but keep your decryption key private

- **How does this work?**
  - Requires some one-way function:
    - Can't (easily) determine the decryption key from the encryption key

  - Depends on computational complexity of factoring large numbers
  - Specifically, take primes p and q: the product pq has only 2 factors!
Cryptography Strength?

- DES vulnerable to cracking in few hours
  - But can overcome this by applying DES 3 times in succession using different keys each time ("Triple DES") providing fairly secure results

- RSA published a 140-digit prime in a 1977 article and offered a $100 reward to whomever could factor it
  - Was thought it would take 40 quadrillion years to factor (based on 1977 technology)
  - Factored in 1994 using a network of about 600 computers from volunteers across the Internet
  - RSA-576 (174 decimal digits) cracked in Dec'2003