Outline

Objective:
- Access Control Mechanisms

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, rm *

Reliability Mechanisms
(Redundancy)

• Replication of data, geographically distributed
  – As simple as backups
  – First-class replication (Coda)
• Error detection-correction
  – Parity bits, checksums
  – Voting schemes

Step 0: Basics of Cryptography

Relationship between the plaintext and the ciphertext
Intrusions

- Monoalphabetic substitution
  - each letter replaced by different letter
- Given the encryption key,
  - easy to find decryption key
- Secret-key crypto called symmetric-key crypto
  - if keys are long enough there are OK algorithms
  - secret key must be shared by both parties
  - DES (Data Encryption Standard)

Secret-Key Cryptography

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
- RSA (Rivest, Shamir, Adelman)
**One-Way Functions**

- Function such that given formula for \( f(x) \)
  - easy to evaluate \( y = f(x) \)
- But given \( y \)
  - computationally infeasible to find \( x \)
- Example: Hash functions – produce fixed size result
  - MD5 – Rivest’s Message Digest Algorithm
  - SHA – US Gov’t Message Digest Algorithm

**Digital Signatures**

- Computing a signature block
  - Hash is fixed length – apply private key
- What the receiver gets
  - Use public key on signature block to get hash back
  - Compute the hash of document part
  - Do these match?
- Assumes \( E(D(x)) = x \) when we usually want \( D(E(x)) = x \)
- Public key must be known by receiver somehow – certificate

**Distributing Public Keys**

- Certificate authority
  - Trusted 3rd party
  - Their public key known
- Send name and public key, digitally signed by CA (certificate authority)
### Step 1: 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 for access control

### Authentication Using Passwords

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

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

• How a cracker broke into LBL
  – a U.S. Dept. of Energy research lab
Authentication Using Passwords

<table>
<thead>
<tr>
<th>Name</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bobbie</td>
<td>4238, e(Dog4238)</td>
</tr>
<tr>
<td>Tony</td>
<td>2918, e(6%%Taedic918)</td>
</tr>
<tr>
<td>Laura</td>
<td>6902, e(Shakespeare6902)</td>
</tr>
<tr>
<td>Mark</td>
<td>1694, e(XaB@Bwcz1694)</td>
</tr>
<tr>
<td>Deborah</td>
<td>1092, e(LordByron, 1092)</td>
</tr>
</tbody>
</table>

The use of salt to defeat precomputation of encrypted passwords

One-Time Passwords

Using 1-way function:
- 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
  - Choose password s and integer n
  - 1st time P1 = f(f(f(s))), 2nd time P2 = f(f(f(s))), etc
  - Login name supplies current integer value
  - Server stores old password, f(newpassword) = old?
**Challenge - Response**

- Sets of question – answer pairs
  - Server picks one and asks
  - User knows answer
- User picks function f(x)
  - Server sends a value for x
  - User sends back f(x) as password
- Using symmetric encryption
  - Server sends random value r
  - User encrypts with secret key – e(r,k)
- Server compares

**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.
- Retinal scans
- Voice recognition
**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

**Step 2: Access Control Mechanisms**

**Protection Domains**

Examples of three protection domains

**The Access Model**

- Authorization problems can be represented abstractly by use of an access model.
  - each row represents a subject/principal/domain
  - each column represents an object
  - each cell: accesses permitted for the \((\text{subject}, \text{object})\) pair
    - read, write, delete, execute, search, control, or any other method
- In real systems, the access matrix is sparse and dynamic.
  - need a flexible, efficient representation
Access Control Matrix

- Processes execute in a protection domain, initially inherited from subject (user running the process)

<table>
<thead>
<tr>
<th></th>
<th>TA</th>
<th>grp</th>
<th>Terry</th>
<th>Lynn</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>proj</td>
<td>rx</td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>t</td>
<td>r</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

File Access Control

- Access control lists - detailed list attached to file of users allowed (denied) access, including kind of access allowed/denied.
- UNIX RWX - owner, group, everyone
- Capabilities – permitted accesses associated with subject (user), similar to an address space.
  - Un-forgable object reference, like a pointer.

Trusted Systems

- Trusted Computing Base

A reference monitor
Access Control Lists

- **Approach**: represent the access matrix by storing its columns with the objects.
  - Tag each object with an access control list (ACL) of authorized subjects/principals.
- To authorize an access requested by $S$ for $O$
  - search $O$'s ACL for an entry matching $S$
  - compare requested access with permitted access
  - access checks are often made only at bind time

Use of access control lists of manage file access

<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
UNIX access control

- Each file carries its access control with it.
- rwx rwx rwx setuid
- Owner UID
- Group GID
- Everybody else
- When bit set, it allows process executing object to assume UID of owner temporarily - enter owner domain (rights amplification)
- Owner has chmod, chgrp rights (granting, revoking)

Capabilities

- Approach: represent the access matrix by storing its rows with the subjects.
- Tag each subject with a list of capabilities for the objects it is permitted to access.
  - A capability is an unforgeable object reference, like a pointer.
  - It endows the holder with permission to operate on the object
    - e.g., permission to invoke specific methods
    - Typically, capabilities may be passed from one subject to another.
  - Rights propagation and confinement problems

Capabilities

Each process has a capability list
Capabilities

- 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

Calling Mechanism

- Assume a capability to execute a procedure (lpr)
- Parameter is a capability to an object of mine (read access to file foo)
- lpr’s c-list has a template capability for the parameter, requires that certain rights are passed in (if lpr can’t read the file foo, it obviously can’t do its job)
- Sometimes, need rights amplification to do the job.

Dynamics of Protection Schemes

- How to endow software modules with appropriate privilege?
  - What mechanism exists to bind principals with subjects?
    - e.g., setuid syscall, setuid bit
  - What principals should a software module bind to?
    - privilege of creator: but may not be sufficient to perform the service
    - privilege of owner or system: dangerous
Dynamics of Protection Schemes

Problems:

- **Mutual Suspicion**
  When a caller requests a service, the callee should be limited in what stuff of the caller's it has access to (only what it is passed). (entering domain - setuid vs. passing in single object)

- **Modification**
  When caller does pass an object as parameter, we want to control whether the callee can modify it (I send you a photo, you remove my copyright mark)

- **Conservation**
  When caller passes an object to gain service, we want to know that the callee doesn't retain or even pass on the right to the object after we think they are done with it (our credit card number and selling such databases)

- **Confinement**
  We want to prevent information leaks (besides the objects themselves)

---

Dynamics of Protection Schemes

- How to revoke privileges?
- What about adding new subjects or new objects?
- How to dynamically change the set of objects accessible (or vulnerable) to different processes run by the same user?
  - Need-to-know principle / Principle of minimal privilege
  - How do subjects change identity to execute a more privileged module?
    - protection domain, protection domain switch: (enter)

---

Protection Domains

- Processes execute in a protection domain, initially inherited from subject
- Goal: to be able to change protection domains
- Introduce a level of indirection
- Domains become protected objects with operations defined on them: owner, copy, control

<table>
<thead>
<tr>
<th></th>
<th>grant/s</th>
<th>protection</th>
<th>proj</th>
<th>ln/vr</th>
<th>hog/svp</th>
<th>Domain0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>rw</td>
<td>rw</td>
<td>r</td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grp</td>
<td>r</td>
<td>rw</td>
<td>r</td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terry</td>
<td></td>
<td>r</td>
<td>r</td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lynn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• If domain contains copy on right to some object, then it can transfer that right to the object to another domain.
• If domain is owner of some object, it can grant that right to the object, with or without copy to another domain.
• If domain is owner or has ctl right to a domain, it can remove right to object from that domain.
• Rights propagation.

<table>
<thead>
<tr>
<th>TA</th>
<th>grp</th>
<th>Terry</th>
<th>Lynn</th>
<th>Domain0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>r</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>rwx</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>read</td>
<td>write</td>
<td>read</td>
<td>write</td>
<td>read</td>
</tr>
</tbody>
</table>

Dynamics of Protection Schemes

Problems:
• Mutual Suspicion
  enter my domain vs. me, as owner, granting right to another domain (temp)
• Modification
  rights for particular operations
• Conservation
  ctl – being able to revoke rights; not granting copy rights
• Confinement
  remove rights to write or create anything else?

NTFS File Encryption

Operation of the encrypting file system

Random 128-bit key, \( K \)

\( K \) retrieved by applying user’s private key to stored key on disk

\( K \) encrusted with user’s public key

\( K \) encrusted with encrypted file

\( C \) = Encrypted file

Disk

Modified DES

Plaintext file

Encryption

K

Modified DES

Modified DES

K

Plaintext file

Decryption

Encrypted file

K retrieved by applying user’s private key to stored key on disk

K encrusted with user’s private key

K encrusted with user’s public key

Operation of the encrypting file system