Security

At least three different categories:

- **Data integrity**: Backup etc., taken care by normal OS functions
- **Protection against user errors**: done by separating users and processes
- **Protection against malicious users**: more complicated, involves trade-offs between ease of use and level of protection

Major problem: **Identification of users**
Most common scheme: Passwords
+ : Easy to use and understand
- : All too often too easily guessable
- : Exposure problem
- : Where to store passwords

Other problems arise from vicious programs. Most famous example so far: **Internet Worm** (1988)
Very clever use of weaknesses in OS design/program bugs

Protection mechanisms

Access to shared resources must be controlled

Two aims:
- Protection against users’ mistakes
- Increase in reliability

Principles:

- separate policy (**what**) from mechanism (**how**)
  Important for flexibility
- Users should have as much privilege as necessary to get job done

Standard way: each user separate domain of protection
Need **trusted way** of making system privileges available
Disadvantage: primary target for breakins (**setuid**-programs in UNIX)

Access matrix

abstract view of protections
have row for each domain and column for object
Entry indicates access right
Example:

<table>
<thead>
<tr>
<th></th>
<th>F_1</th>
<th>F_2</th>
<th>F_3</th>
<th>Printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_1</td>
<td>read</td>
<td>write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D_2</td>
<td></td>
<td></td>
<td>print</td>
<td></td>
</tr>
</tbody>
</table>

Have capabilities like read, write, copy, owner, control

Unsuitable for implementation because matrix far too large

Implementation Issues

- Access lists for objects (store columns)
- Capability lists for domains (store rows)

Capabilities allow great flexibility
Example: **Hydra**

- Auxiliary rights: each process can pass access to procedure to other processes ⇒ dynamic change of access rights
- Rights amplifications: procedure can act on specified type on behalf of any process which is allowed to execute it
  Rights cannot be passed on
  ⇒ flexible and more secure mechanism for granting higher privileges temporarily
UNIX access rights

coarser than access lists

for each file, have three categories of possible users

- owner
- group (pre-defined set of users)
- all others

owner can grant permission to

- read
- write
- execute program/find files in directory

Authentication in Networks

Aim: authentication in insecure networks

Two possible solutions:
- Central authentication server (Kerberos):

Have following protocol:

- Client → Auth Server: Credentials, please?
- Auth Server → Client: ((Client Id, Session Key)\textsubscript{S}, Session Key)\textsubscript{C}
- Client decrypts message and keeps Session Key
- Client → Server: (Client Id, Session Key)\textsubscript{S}
- Server decrypts message and obtains Client Id and Session Key

Authentication Server must be trusted
Protocol vulnerable to replay attacks

- Newer Alternative: PGP (Pretty Good Privacy)

Relies on so-called Public Key Cryptography:

Have two different keys, public key \( K_p \) and secret key \( K_s \)

Clear text encrypted via \( K_p \) can be decrypted only via \( K_s \) and vice versa

Assume every host in network has public and secret key

Now communication can take place by sending message from \( A \) to \( B \) as \( (m_A)_{B_p} \). Only \( B \) can decode the message, and it can verify that it came from \( A \).

Certification of keys still necessary.