Yesterday, we had this diagram of an access matrix:

```
  obj1  obj2  obj3  obj4
---  ---  ---  ---
user1 r   -    -    r
user2 -   r    -    -
user3 r   rw   rw   -
user4 -   -    -    *
```

Unix actions: rwx

Windows also adds "delete", "read", "write", permissions, along: "sys", "sync".

Windows also adds a "Discretionary Act" to each object. Violations are parallel, but logged.

User would:

- This is the simplest situation
- Can't be handled
- Unix has that can't be handled
- User/group/username

Here's an Access Control List for this object:

- Unix has
- That can't be handled
- RW
- Allow
-RW
- Allow
- *
- Deny

setuid - special Unix security feature for privilege escalation

No matter who invokes the setuid executable, it runs as the owner.

On old systems, saving mail involved directly editing the system mail spool, so
the mail program would be setuid. (Also, su and sudo are setuid, because
you must be root to become root - it's not used for too much else these days)

There's a big security hole if a setuid program can be tricked into doing
the wrong thing. (Nowadays, setuid programs have largely been replaced
by daemons/ servers of various sorts, since they're accessible from off-system)

Least Privilege (this is why you don't log in as root, even if you have the
root password). On Unix, every process has a uid (the id of the user it's
running as). Example: an FTP daemon that allows users to log in needs to
be root so it can do file operations as those users. Dangerous. But that's
dangerous. So if it uses setuid() to become the lower-privileged user, (In order
that it can go back, the system stores a "saved uid" in addition to the
"effective uid" that setuid() changes. The real, "saved" uid is only looked at to
check to see if setuid is legal.

OS Privileges (POSIX "capabilities")

Who can shutdown, install drivers, set the system time, create device, etc...?

We'd like to write daemons that only use the privileges that they actually need.
Real Capabilities (theorized about a lot, but not widely used)

A permission is a map object ➔ user, action
A capability is a map user ➔ object, action (both create access matrices when you expand them out)

You could regard Kerberos as a capability system of capabilities

In many implementations, a ticket is "just a bit pattern." The user can copy and give it away. But the capability could be held by the OS, which could restrict physical security of the client.