Computer Security

Goal

To learn the basics of how data on computers is protected.

Security and Protection

Goals: confidentiality, integrity, availability
Requirements vary from site to site; describe:
- *Orange book* (DoD Trusted Computer System Evaluation Criteria)
- *Red Book* (DoD Trusted Network Interpretation of the TCSEC)
- *Privacy Act*

all of which constrain policies at different sites

Discuss *policy* vs. *mechanism*.
Our job: design mechanisms that will allow these goals to be accomplished, or think about why such mechanisms are infeasible

Saltzer's and Schroeder's Design Principles

Following these is central to any secure architecture

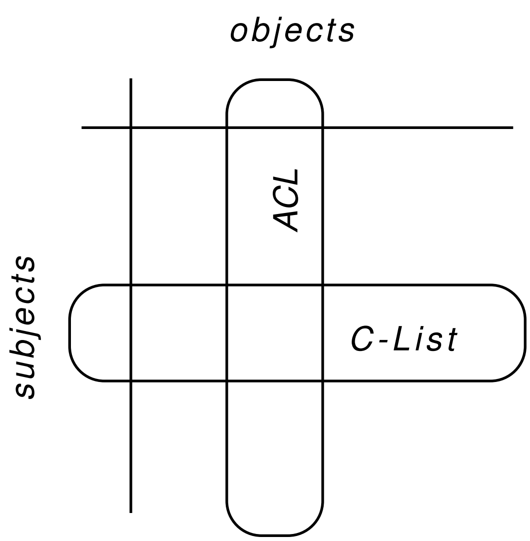
- *Economy of Mechanism*: the protection mechanism should have a simple and small design.
- *Fail-safe Defaults*: the protection mechanism should deny access by default, and grant access only when explicit permission exists.
- *Complete Mediation*: the protection mechanism should check every access to every object.
- *Open Design*: the protection mechanism should not depend on attackers being ignorant of its design to succeed. It may however be based on the attacker's ignorance of specific information such as passwords or cipher keys.
- *Separation of Privilege*: the protection mechanism should grant access based on more than one piece of information.
- *Least Privilege*: the protection mechanism should force every process to operate with the minimum privileges needed to perform its task.
- *Least Common Mechanism*: the protection mechanism should be shared as little as possible among users.
- *Psychological Acceptability*: the protection mechanism should be easy to use (at least as easy as not using it).

Issues

Physical security covers access to the computer and associated equipment; controlled by badges, etc.

Operational security covers polcies and procedures used to run the system, and includes:

- access control: who can do what to each object
  subjects: actors
  objects: either passive entities or any entity (subjects too)
  protection domain: what objects a subject can access, and how.
  The basic structure is an *access control matrix* :



Access control lists (ACLS) are the columns of the access control matrix.
Capability lists (C-Lists) are the rows of the access control matrix

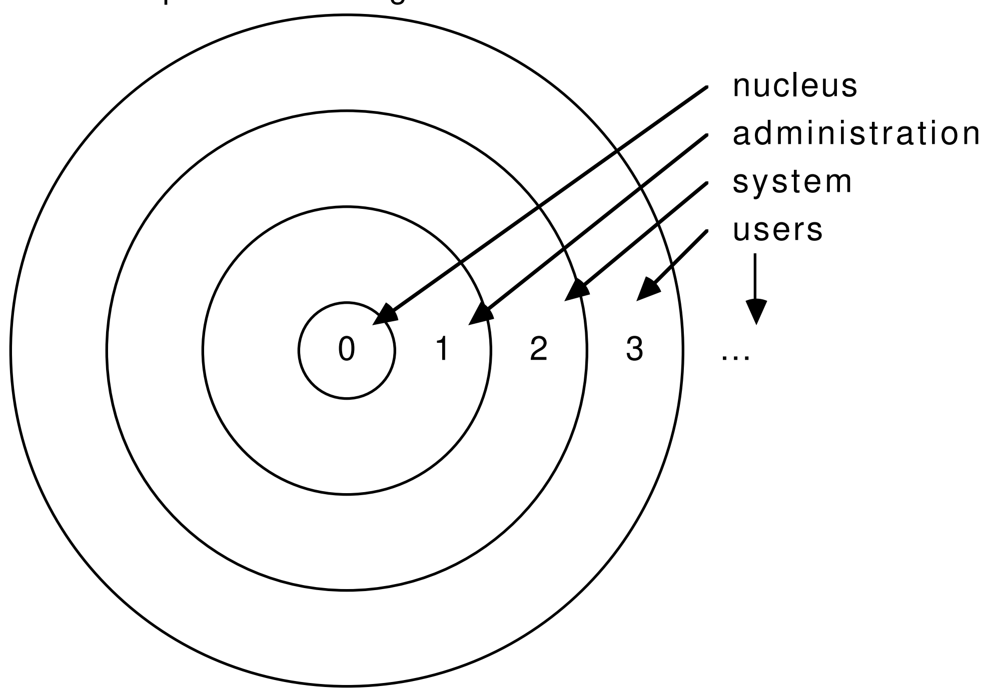
ACLs go down; C-lists across.
Discuss capabilities:

- the main difference is a capability is thought of as the only means to refer to an object
- capability-based addressing: although we've talked about C-lists being for secondary storage (files, etc.) they are also used in main memory. Compare capability lists to segment tables:
  - both consist of pointers and rights
  - difference: entries in a segment table point to in-core structures (segments), but entries in a c-list point to secondary storage .

  So combine them: use capabilities for both in-core and secondary storage objects. That is, associate a unique capability with any object, regardless of where it is in system; the same one is used by the file manager and the memory manager.

global object table enables this. Each entry in the table points to exactly 1 object regardless of where it is; if the object moves, the table is updated. A capability is an index into this table, augmented with a set of rights.
problem: must prevent users from creating capabilities. This is done in one of two ways:
(1) store the capability in segments which the user's can't access
(2) use a tagged architecture in which each main store location has a tag bit; if the tag bit is set, only system routines can minipulate that location. So put the capability there.

example: MULTICS implements access control using access control lists in a dynamic linking environment based upon concentric protection rings:

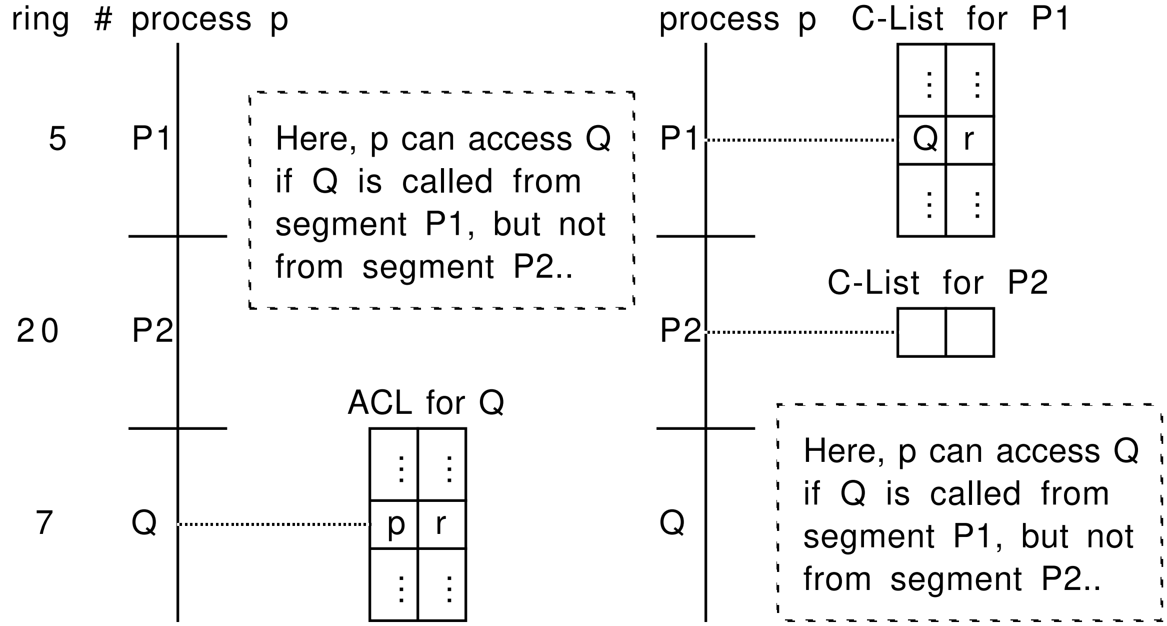


Each segment is assigned to a ring; the assignment is fixed, and is part of the entry in the file/directory for each segment {in practice, there is actually a bracket}

A process can only access going out; *ie*, if a process executing in segment S in ring i tries to access segment T in ring j, the access is allowed if:
(1) $j \geq i$; and
(2) ACL for T allows the access

If $i > j$, an interrupt occurs, and control is passed to the system routine to verify the validity of the access.

Note even if a process is going out, it must copy arguments, addresses of variables in ring i, *etc.*; these copies must be put into an area accessible to T. This means …
problem: objects must be linearly ordered
example: with capabilities, this is not true:



Other interesting note: with capabilities, you do not need privileged hardware states. Just take an object-oriented approach: to do something, you need an object (thing to be manipulated) and an operation. The object is of a certain type.
example: consider scheduling and dispatching (should **only** be done by the kernel). So, "scheduling" and "dispatching" are operations defined for process objects, and undefined for all other objects.
Just make sure that no user gets capabilities for the objects which are of the types on which privileged operations work.
problem: Why are capabilities not used more often? Too expensive, and such systems execute more slowly

- logging actions and examining system state (*logging and auditing*)
  - good mechanism for detecting (potential) problems
  - useful with IDES and other surveillance programs;

  requires …
- verification of user identity (*user authentication*)
  - passwords, keys, and/or biometrics
  - limits on number of attempts to log in

  leading to …
- keeping data private (*confidentiality*)

- keeping data unaltered (*integrity*)

Cryptography

Used as a tool to augment (or replace) other security mechanisms, or where other security mechanisms infeasible (*ie*, packet radio networks). Two goals:

Privacy is protection against disclosure.

Authentication is assurance of identity.

Typical uses are to provide privacy to personal or proprietary information, and to provide assurance of its source.

Two styles:

Classical cryptography uses one secret key; privacy is done by keeping the key and message secret, whereas authentication is done by keeping the key secret but making the message public.

Public Key cryptography uses two keys, one public, one private; privacy is done by encrypting using the recipient's public key (which she can decrypt using her private key), and authenticity is done by encrypting using the sender's private key (which anyone can decrypt using her public key).

Attacks on cipher systems:

(1) *Ciphertext Only* attack: given ciphertext, determine message (and key, if possible)

(2) *Corresponding Plaintext and Ciphertext* attack: given ciphertext and corresponding plaintext, determine key

(3) *Chosen Plaintext* attack: obtain ciphertext corresponding to any plaintext you choose, and then find the key.

Classical Ciphers

Review briefly the concepts of types of ciphers, Cæsar ciphers, etc. here.

Public Key Ciphers

Requirements: let D be private key, E public; m message, c ciphertext

(1) $DEm = m$

(2) Given E, it is “computationally infeasible” to get D

(3) E cannot be determined by a chosen plaintext attack

Best known is:

RSA Cryptosystem

(1) Pick two large prime numbers, p and q

(2) Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$ (Euler's totient function)

(3) choose any d such that gcd (d, ƒ(n)) = 1

(4) find e such that ed = 1 mod ƒ(n)

Here, the public key is (e, n), and the private key is d. Given n, it is computationally infeasible to find $\phi(n)$ because the techniques known all involve factoring n (or doing the equivalent).

To encrypt a (plaintext) message m, compute

$$c = m^e \bmod n$$

and to decrypt ciphertext c, compute

$$m = c^d \bmod n$$

This works because of Euler's Generalization to Fermat's Little Theorem, which says that

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

when gcd(a, n) = 1; so

$$m = c^d \bmod n = (m^e \bmod n)^d \bmod n = m^{ed} \bmod n = m^{k\phi(n)+1} \bmod n$$
$$= ((m^{\phi(n)} \bmod n)^k \bmod n)\ (m \bmod n) = m \bmod n = m$$

This is considered a very strong cipher provided you pick n to be at least 512 bits; factoring techniques will make lesser values soon factorable.

example: We want to use the encoding a = 00, …, z = 25, <blank> = 26, and write letters in pairs; this means n must be at least 2626 (so we can represent all possible pairs of letters). Take p = 53, q = 61; then n = 53 $\times$ 61 = 3233, and $\phi(n)$ = 52 $\times$ 60 = 3120. Pick d = 71 (as gcd(71, 3120) = 1); this gives e = 791. Note

$$de \bmod \phi(n) = (71)(791) \bmod 3120 = 1.$$

So, the public key corresponding to the private key 791 is (71, 3120). Now, to encrypt "renaissance" for this user, we split it into pairs of letters:

re na is sa nc e<blank>

In the encoding, this becomes

1704 1300 0818 1800 1302 0426

To encrypt for privacy, we use the recipient's public key, so the ciphertext is

3106 0100 0931 2691 1984 2927

as $1704^{71} \bmod 3233 = 3106$, *etc*.

SIgnatures (for authenticity): Suppose our private key is 61; then our public key must be (1381, 3233). So to sign the message "renaissance", again we get the encoding:

1704 1300 0818 1800 1302 0426

and encrypt it but using our private key:

2436 0629 0818 0336 1302 1890

as $1704^{61} \bmod 3233 = 2436$, *etc*. Note that anyone can read this (as the decryption key is public), but without the private key no-one could generate it!

One-Way Functions

Describe these: noninvertible functions (like a good hash function); require:

(1) Given a message m, easy to compute c = f(m);

(2) Given a ciphertext c, computationally infeasible to find any m such that c = f(m)

(3) Given a ciphertext c and its corresponding message m, computationally infeasible to find an m' such that c = f(m') and m' ≠ m.

example: UNIX password encryption algorithm

Describe it, including salt