Security and Protection

Security is a tougher problem in distributed systems:

- resources and accessing agent may reside on different machines
- may not have central authority
- may not want to trust other kernels
- migration

Attacks on security

Active attacks:

- **browsing:** non-detructive examination of all accessible data

⇒ protection system

- **leaking:** obtain information via (authorised) accomplice

⇒ *confinement problem*

- **infering:** traffic analysis, code breaking, ...

⇒ ???

- **masquerading:** *trojan horses*, *spoofing*, ...

⇒ ???

Protection mechanisms cannot solve all problems,
users must be security-conscious too!

Attacks on security...

Passive attacks:

- **virus:** self-replicating program embedded in some host

⇒ paranoia

- **worm:** program which replicates across network

⇒ network security

- **logic bomb:** malicious program (in T.H. or virus) triggerd by some event

⇒ see trojan horse, virus

- communications-based attacks...

Attacks on security...

Communication-based attacks:

- **integrity:** intruder changes message
- **authenticity:** intruder illegally connects as a "valid" network client
- **denial:** intruder fully or partially prevents communication between valid users
- **delay:** intruder delays delivery (which may render service useless)
- **replay:** intruder retransmits old messages

Security

Can divide security issues in two categories:

- resource protection
  similar to, but more difficult than in centralised system
- communication security
  particular to distributed system
  requires
  - physically secure network, or
  - encryption (covered later)

Resource protection

... also has two aspects to it:

- establishing whether the attempted access is within agent's rights
  same as in centralised system
- establishing whether agent's identity is genuine
  harder across network

⇒ *authentication* problem
  Can authenticate
  - user agent
  - kernel

  Authentication requires cryptographic methods (covered later)

Access matrix model

	Objects			
Agents	S_1	S_2	O_3	O_4
S_1	terminate	wait, signal, send	read	
S_2	wait, signal, terminate			read, execute write
S_3		wait, signal, receive		
S_4	control		execute	write

Used by protection system to determine whether access is allowed

Access matrix properties

- rows define agents' *protection domains*
- columns define objects' *accessibility*
- dynamic data structure: frequently
  - permanent changes (e.g. `chmod`)
  - temporary changes (e.g. `setuid`)
- Access matrix is very *sparse* and has a lot of repeated entries

⇒ usually not stored explicitly

Issues

- propagating rights
- restricting rights
- amplifying rights
- revoking rights

Access matrix implementation: ACLs

Can represent access matrix column-wise: *access control list* (ACL)

+ easy to determine who has access
+ easy to revoke access
– difficult to determine agent's protection domain
• often simplified by introduction of *domain classes*
  e.g. UNIX groups
+ can have *negative rights* to reduce "window of vulnerability"

Used by Andrew File System, Windows-NT, etc

Access matrix implementation: Capabilities

Can represent access matrix row-wise: *capabilities*

+ protection domain = {*capabilities*}
– accessibility hard to determine
– revocation difficult
+ don't need to trust intermediary

Can also have combination: *Lock-key* model:
access allowed only if ACL (lock) and capability (key) match

Problems with access matrix model

Cannot prevent one (authorised) agent from (intentionally or not) accessing a resource on behalf of another (unauthorised) agent
⇒ *confinement problem*

Information can be leaked via

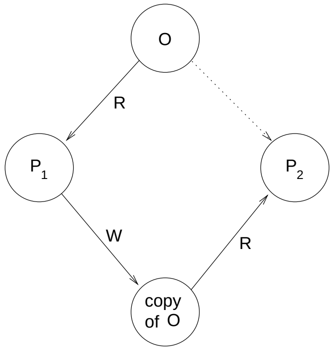
storage channels e.g. via files

legitimate channels e.g. during normal communication

covert channels e.g. by modulating resource usage
in general impossible to prevent

Only way out: *limit information flow*

Information flow models



Can only guarantee confinement of O's data by not allowing *any* information to flow from O to P_2, directly or indirectly

Denning's Lattice Security model

Models military classification system:

- each object O has *classification* $C(O)$ (unclassified, confidential, etc.) denoting security level
- each agent A has a *clearance* $C(A)$ to access certain security levels
- security levels are *partially ordered*
- each object & agent is affiliated with a *compartment* K
- access of O by A is allowed iff

  $C(A) \geq C(O)$ *sufficient clearance*, **and**
  $K(A) \subseteq K(O)$ *need to know*

Note: "Need-to-know" is a general principle used in other models too

Bell & LaPadula's flow-secure access control

- combine access matrix with security clearance system
- operations: read-only, append, execute, read-write, and control

read	$C(A) \geq C(O)$
append	$C(A) \leq C(O)$
read-write	$C(A) = C(O)$

- ensure that information can only flow "upwards"
  Ref: R. Sandhu: "Lattice-Based Access Control Models", IEEE Computer, Nov 93
  also discusses "Chinese Wall Policy", used commercially (e.g. in Banks to prevent insider trading).

Capabilities

Unforgeable token, possession of which implies some access

Also present **unique names** for objects

tagged ⇒ protected by hardware (most hist. syst., IBM AS/400)

- \+ safe (in centralised systems)
- – requires special hardware, problematic for distribution???

partitioned ⇒ protected by system software (Mach, Grasshopper)

- \+ know who has them, revocation easy
- – system must marshal on IPC

sparse ⇒ protected by obscurity (Amoeba, Password Capability System)

- \+ like data, can easily passed in messages etc.
- \+ no need to trust remote kernel
- – don't know who has them, revocation difficult

Partitioned capabilities

- stored in system-protected *capability lists* (clists)
- user code references capabilities indirectly (clist index)
- handing capabilities to other agents requires system intervention
- message passing requires headers only accessible to systems for transmitting capabilities (as in Mach)
- allow easy reference counting

Sparse capabilities

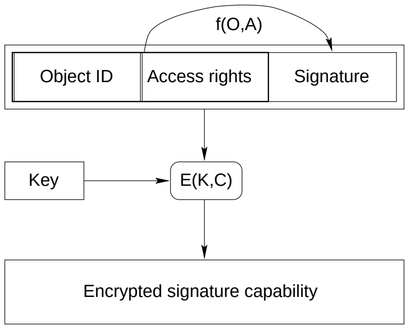
Consist of:

- object ID
- access rights (optional)
- "random" bit string for sparsity:
  - encrypted info about object, or
  - password

How can they be made safe?

Signature capabilities

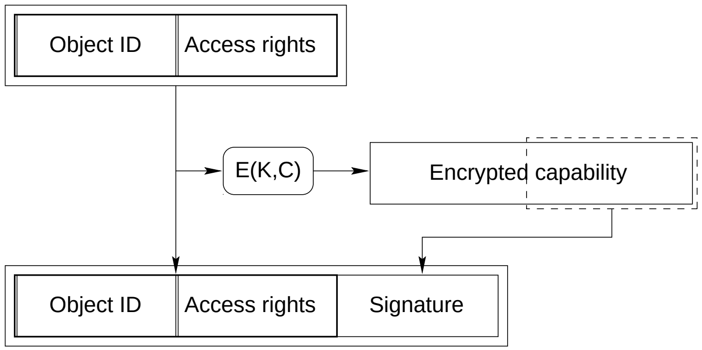
"First Migration Scheme" [Gligor, Lindsay, IEEE T-SE 1979, p 607]



+ tamper proof via encryption with secret kernel key
+ can freely be passed around
− need to decrypt on each validation
− users do not know which object capability refers to

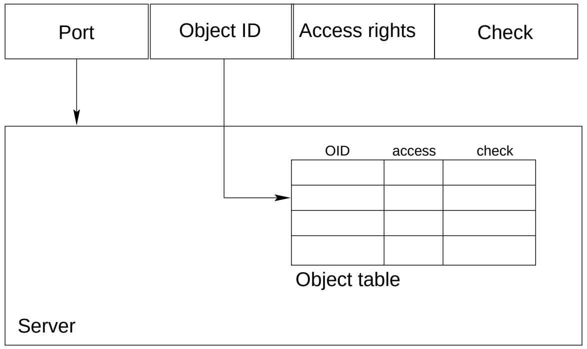
Signature capabilities

"Second Migration Scheme" [Gligor, Lindsay 1979]



makes object ID visible, yet still tamper proof

Amoeba's capabilities



S.J. Mullender and A.S. Tanenbaum: The design of a capability-based distributed operating system. *The Comp. J.*, 29:289–299, 1986.

Properties

- port identifies server (kernel caches server location)
- port IDs are large (48-bit) sparse numbers
  knowing port implies send rights
- server uses OID to look up rights, check fields to compare

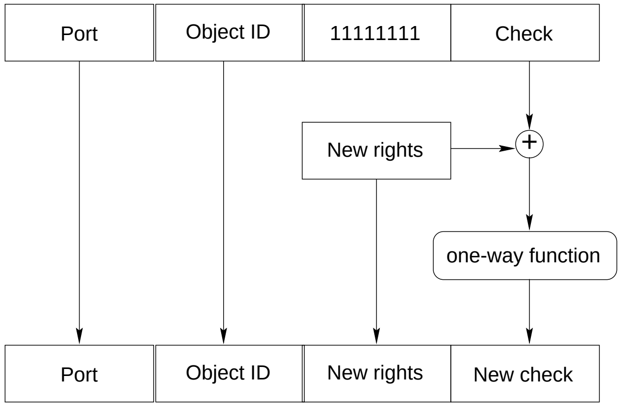
+ tamper proof, can be passed around freely

- original ("owner") capability has all rights

+ server can derive lesser capabilities

- revocation requires asking server to change check field
  ⇒ revokes access for all holders

Rights restriction



Server can derive lesser capabilities on request

Improved version (not implemented)

- set of *commuting* one-way functions f_i, one for each access mode
  $f_i(f_j(x))) = f_j(f_i(x))$
- to remove access mode i, obtain new check field as
  $C' = f_i(C)$
- can be done by user without server intervention

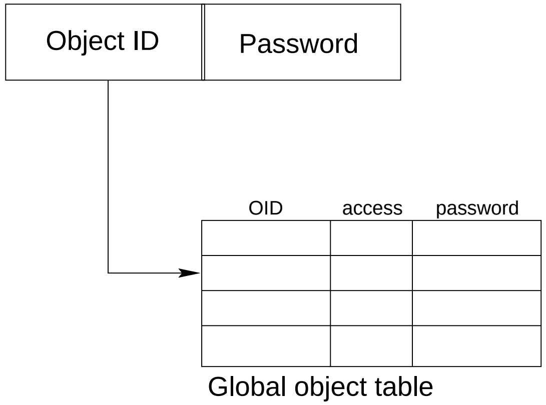
Server authentication: F-boxes

- hardware device "F-box" at each network connection
- when requesting messages for port G, F-box will only accept messages destined for port $P = f(G)$, where f is a one-way function
- server publishes P as port ID
- intruder who does not know G cannot access messages

Scheme depends on physical security of F-boxes (or their implementation in the OS)

Never been implemented (to my knowledge).

Password capabilities



M. Anderson, R. Pose, C.S. Wallace: A password-capability system. *The Comp. J.*, 29:1–8, 1986.

Properties

- no encryption ⇒ easy to validate
- password must be hard to guess
  ⇒ need encryption scheme or H/W for generation
- need separate passwords for different rights

⇒ good idea to put rights field in capability (similar to Amoeba)

- OID + null password is a valid capability conferring no rights

⇒ useful for implementing ADTs (system enforced encapsulation)

- location independent
- passwords need protection (eavesdropping)
- selective revocation possible if allow multiple passwords

Rights amplification

- give user temporarily increased privileges (a la UNIX setuid)
- typically done by
  - having server perform privileged operation on behalf of client
  - client and server communicate via RPC
- Capability systems can use special mode indicating certain privileges
  - AS/400 *profile adoption/propagation*:
    invocation of object merges caller's PD with (subset of) object's owner's [V. Berstis: Security and protection in the IBM System/38. In *Proc. 7th Symp. Comp. Arch.*, pages 245–250. ACM/IEEE, 1980.]
  - Mungi *protection domain extension* (PDX):
    controlled temporary extension of caller's PD
    by a PD registered for object [J. Vochteloo et al, 1996]

[1] Ravi S. Sandhu. Lattice-based access control models. *IEEE Comp.*, 26(11):9–19, 1993.

[2] Viktors Berstis. Security and protection in the IBM System/38. In *Proc. 7th Symp. Comp. Arch.*, pages 245–250. ACM/IEEE, May 1980.

[3] Frank G. Soltis. *Inside the AS/400*. Duke Press, Loveland, CO, USA, 1996.

[4] Richard Rashid, Avadis Tevanian, Jr., Michael Young, David Golub, Robert Baron, David Black, William J. Bolosky, and Jonathan Chew. Machine-independent virtual memory management for paged uniprocessor and multiprocessor architectures. *IEEE Trans. Computers*, C-37:896–908, 1988.

[5] Alan Dearle, Rex di Bona, James Farrow, Frans Henskens, Anders Lindström, and Francis Vaughan. Grasshopper: An orthogonally persistent operating system. *Comput. Syst.*, 7(3):289–312, 1994.

[6] Sape J. Mullender and Andrew S. Tanenbaum. The design of a

capability-based distributed operating system. *The Comp. J.*, 29:289–299, 1986.

[7] M. Anderson, Ronald Pose, and Chris S. Wallace. A password-capability system. *The Comp. J.*, 29:1–8, 1986.

[8] V.D. Gligor and B.G. Lindsay. Object migration and authentication. *IEEE Trans. Softw. Engin.*, 5:607–611, 1979.

[9] Jerry Vochteloo, Kevin Elphinstone, Stephen Russell, and Gernot Heiser. Protection domain extensions in Mungi. In *Proc. 5th Int. W. Obj. Orient. Operating Syst.*, pages 161–165, Seattle, WA, USA, October 1996. IEEE.

