PROTECTION MECHANISMS

Protection: set of mechanisms to ensure security of a computer system

TWO BASIC SECURITY ISSUES:

➜ **internal security (access permission):**
Establishing legality of an agent's access to some object.

➜ **External security (authentication):**
Establishing agent's identity:

- Login authentication
- Authentication of remote agents (requires cryptographic methods)

Here we concentrate on the issue of access rights.

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: frequent
  - ➜ permanent changes (e.g. chmod)
  - ➜ temporary changes (e.g. setuid)
- Very *sparse* with many repeated entries
- Usually not stored explicitly.

ISSUES FOR PROTECTION SYSTEM DESIGN

- Propagation of rights:
  - ➜ Can agent grant access to another?
- Restriction of rights:
  - ➜ Can agent propagate restricted rights?
- Revocation of rights:
  - ➜ Can access, once granted, be revoked?
- Amplification of rights:
  - ➜ Can unprivileged agent perform restricted operations?
- Determination of object accessibility:
  - ➜ Which agents have access?
  - ➜ Is object accessible at all (garbage collection)?
- Determination of agent's protection domain:
  - ➜ Which objects are accessible?

ACCESS MATRIX IMPLEMENTATION: ACLS

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

- ➜ ACL associated with object.
  - ➜ Propagation: meta-right (e.g., *owner* can `chmod`)
  - ➜ Restriction: meta-right
  - ➜ Revocation: meta-right
  - ➜ Amplification: protected-invocation right (e.g., `setuid`)
  - ➜ Accessibility: explicit in ACL
  - ➜ Protection domain: hard (if not impossible)
- ➜ Usually condensed by introduction of *domain classes* (UNIX groups).
- ➜ Can have *negative rights*, to:
  - ➜ reduce "window of vulnerability",
  - ➜ simplify exclusion from groups.
- ➜ Sometimes implicit (process hierarchy).
- ➜ Full ACLs used by Multics, Apollo Domain, Andrew File System, Windows-NT.
- ➜ Simplified ACLs used by Unix, VMS, many others...

ACCESS MATRIX IMPLEMENTATION: CAPABILITIES

Can represent access matrix row-wise: *capabilities*

- ➜ *Capability list* associated with agent.
- ➜ Each capability confers a certain right to its holder.
  - ➜ Propagation: copy capabilities between agents (how?)
  - ➜ Restriction: lesser rights require new ("derived") capabilities (how?)
  - ➜ Revocation: requires invalidation of capabilities from *all agents* (how?)
  - ➜ Amplification: special invocation capability.
  - ➜ Accessibility: requires inspection of all capability lists (how?)
  - ➜ Protection domain: explicit in capability list.
- ➜ Usually condensed by introduction of *domain classes* (UNIX groups).
- ➜ Can have *negative rights*, to:
  - ➜ reduce "window of vulnerability",
  - ➜ simplify management of groups of capabilities.
- ➜ Only successful commercial capability system: IBM System/38, AS/400 [Sol97]
- ➜ Popular among research distributed OS.

CAPABILITIES

- Main advantage of capabilities is the fine-grain access control:
  - ➜ Easy to provide specific access to selected agents.
- Capability presents *prima facie* evidence of the *right to access*:
  - ➜ capability ⇒ *object identifier* (naming),
  - ➜ capability ⇒ (set of) *access rights*,
  - ⇒ Any representation must contain object ID and access rights.
  - ⇒ Any representation must protect capability from forgery.
- How implemented and protected?
  - ➜ **tagged** (protected by hardware),
  - ➜ **partitioned** (protected by software),
  - ➜ **sparse** (protected by obscurity).

TAGGED CAPABILITIES

Additional *tag bit(s)* with every (group of) memory word(s).

- ➜ Tags identify capabilities.
- ➜ Capabilities are used like "normal" pointers.
- ➜ Hardware checks permissions on dereferencing capability.
- ➜ User code can copy capabilities.
- ➜ Modifications turn tags off.
- ➜ Only privileged instructions (kernel) can turn tags on.
  - ➜ Propagation easy.
  - ➜ Restriction requires kernel to make new capability.
  - ➜ Revocation virtually impossible (would require memory scan).
  - ➜ Amplification possible (see below).
  - ➜ Accessibility impossible to determine.
  - ➜ Protection domain difficult to establish.

Used in IBM System/38 [Ber80], AS/400 [Sol97], many historical systems.

Protected procedure calls with tagged capabilities (AS/400):

- AS/400 has a segmented memory architecture.
- Capabilities confer rights over segments.
- Capabilities can confer invocation rights.
- Each user has a *profile*, which is essentially a capability list.
- Capabilities can be of *profile adoption* type:
  - ➜ On invocation, segment owner's *profile* is added to caller's protection domain.
  - ➜ Normal pointers can be dereferenced if the profile contains appropriate capabilities.
  - ➜ On return, profile adoption is cancelled.
  - ➜ User can denote subset of their profile to be used in adoption (*profile propagation*).

Tagged capabilities outside RAM:

- Disk has no tags.
- AS/400 page size is 4kB.
- Physical disk blocks are 520B, logical blocks 512B.
- Extra 64B per page store tag bits (among others).
  - ➜ On page-out page must be scanned and all tags collected.
  - ➜ On page-in all tags must be reconsituted.
  - ➜ Significant processing overhead with all I/O.

Tagged capabilities summary:

- Secure through hardware protection.
- Convenient for applications (appear as "normal" pointers).
- Checked by hardware $\Rightarrow$ fast validation.
- Hardware solution is not for everyone.
- Capability hardware is complex (and slow?)
- Separate mechanisms required for I/O and distribution.

Partitioned Capabilities

System maintains capability list (clist) with each process (in PCB).

- User code uses indirect references to capabilities (clist index).
- System validates access via clist when mapping any page.
  - ➜ Validation is implicit at page fault or explicit mapping time.
  - ➜ Propagation requires system intervention to copy between clists.
  - ➜ Restriction requires kernel to make new capability.
  - ➜ Revocation requires kernel to remove cap from all (or specific) clists
  - ➜ Amplification possible (see below).
  - ➜ Accessibility can only be determined by scanning all clists.
  - ➜ Protection domain is explicitly represented in clist.

Used in Hydra [CJ75], Mach [RTY+88], KeyKOS [SSF99], Grasshopper [DdBF+94], Eros [SSF99] and many others.

PROPAGATING PARTITIONED CAPABILITIES (MACH):

- Capabilities can be propagated via IPC.
  1. User must insert capabilities (clist indices) into special field in message.
  2. Kernel looks up clists and inserts representation of “real” capability (*marshaling*).
  3. Receiver’s kernel inserts capabilities into receiver’s clist.
  4. Kernel replaces capability in message by clist index.
- Can be simplified if IPC is local.

Amplification can be performed by schemes similar to AS/400.

PARTITIONED CAPABILITIES SUMMARY:

- Secure through kernel protection.
- Validation at mapping time ⇒ applications can use “normal” pointers.
- Fast validation (clist check is simple, and validation is cached by MMU).
- Propagation requires marshaling and kernel intervention.
- Reference counting possible to detect unaccessible objects.

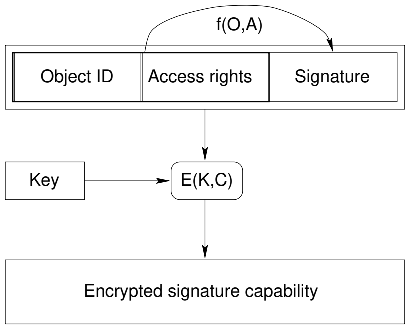
SPARSE CAPABILITIES

Basic idea similar to encryption:

- Add bit-string to make valid capabilities a very small subset of the capability space.
- Can be encrypted object info or something like a password.
- Capabilities are pure user-level objects, which can be passed around like other data.
- Appropriate for user-level servers.

EXAMPLE: SIGNATURE CAPABILITIES:

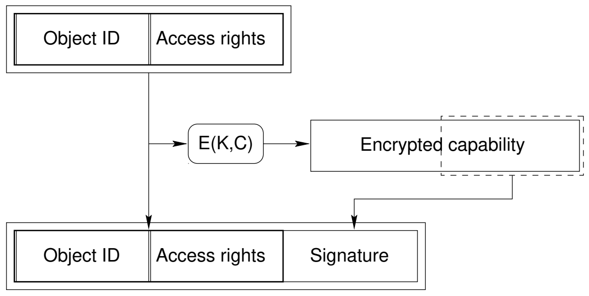
“**First Migration Scheme**” [GL79], designed to allow migration of tagged capabilities in distributed systems.



+ 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

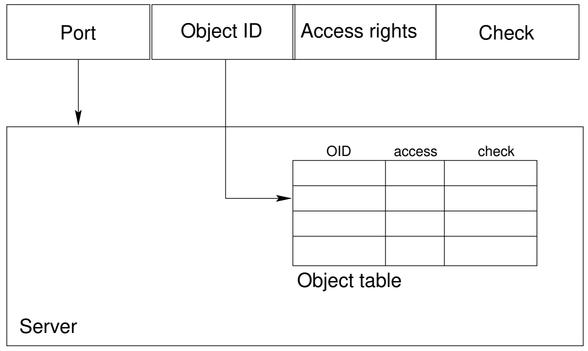
f: one-way function (secure digest), E: encryption function

"**Second Migration Scheme**" [GL79]



Object ID visible, yet still tamper proof.

AMOEBA'S CAPABILITIES:

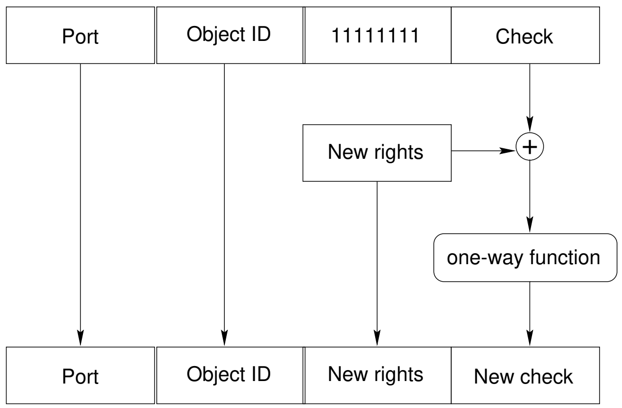


Appropriate for user-level servers [MT86].

PROPERTIES OF AMOEBA CAPABILITIES:

- Port identifies server.
  - ➜ Kernel resolves server and caches server location.
- Port IDs are large (48-bit) sparse numbers.
  - ➜ Knowledge implies send rights.
- Creator ("owner") has all rights.
- Server uses OID to look up rights, checks fields to validate.
  - ➜ Validation done by user-level server when invoked.
  - ➜ Propagation easy, as capabilities are "normal" data.
  - ➜ Restriction requires server to make new capability.
  - ➜ Revocation done by server removing entry from object table.
    But not very helpful if only one capability per access mode.
  - ➜ Amplification possible according to server policies.
  - ➜ Accessibility is impossible to determine.
  - ➜ Protection domain is impossible to determine.

AMOEBA RIGHTS RESTRICTION:



- ➜ Used by server to derive lesser capabilities on request.
- ➜ No need to store derived capability in object table.

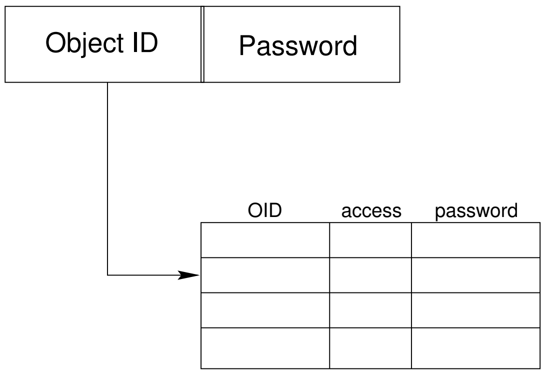
IMPROVED VERSION (NOT IMPLEMENTED):

- Set of *commuting* one-way functions f_i, one for each access mode bit: $f_i(f_j(x))) = f_j(f_i(x))$.
- 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



Global object table

Used in the Monash Password Capability System [APW86], Opal [CLFL94], Mungi [HEV+98].

PROPERTIES OF PASSWORD CAPABILITIES:

- Passwords must be protected (eavesdropping, Trojan horses).
- Separate passwords for different rights (good idea to package rights with caps).
- No encryption $\Rightarrow$ easy to validate.
  - ➜ Validation done by kernel on access or presentation and cached by MMU.
  - ➜ Propagation easy, as capabilities are “normal” data.
  - ➜ Restriction requires kernel to make new capability.
  - ➜ Revocation done by kernel removing entry from object table.
  - ➜ Amplification possible similar to AS/400.
  - ➜ Accessibility is impossible to determine.
  - ➜ Protection domain is known to kernel.

SPARSE CAPABILITIES SUMMARY:

- Statistically secure (like encryption).
- Validation at mapping time ⇒ applications can use “normal” pointers.
- Validation may be slow, but kernel and MMU can cache.
- No kernel intervention required on most operations.
- Reference counting impossible to detect unaccessible objects.

REFERENCES

[APW86] M. Anderson, Ronald Pose, and Chris S. Wallace. A password-capability system. *The Comp. J.*, 29:1–8, 1986.

[Ber80] Viktors Berstis. Security and protection in the IBM System/38. In *Proc. 7th Symp. Comp. Arch.*, pages 245–250. ACM/IEEE, May 1980.

[CJ75] E. Cohen and D. Jefferson. Protection in the HYDRA operating system. In *Proc. 5th ACM Symp. OS Princip.*, pages 141–59, 1975.

[CLFL94] Jeffrey S. Chase, Henry M. Levy, Michael J. Feeley, and Edward D. Lazowska. Sharing and protection in a single-address-space operating system. *ACM Trans. Comp. Syst.*, 12:271–307, November 1994.

[DdBF+94] 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.

[GL79] V.D. Gligor and B.G. Lindsay. Object migration and authentication. *IEEE Trans. Softw. Engin.*, 5:607–611, 1979.

[HEV+98] Gernot Heiser, Kevin Elphinstone, Jerry Vochteloo, Stephen Russell, and Jochen Liedtke. The Mungi single-address-space operating system. *Softw: Pract. & Exp.*, 28(9):901–928, July 1998.

[MT86] Sape J. Mullender and Andrew S. Tanenbaum. The design of a capability-based distributed operating system. *The Comp. J.*, 29:289–299, 1986.

[RTY+88] 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.

[Sol97] Frank G. Soltis. *Inside the AS/400, Featuring the AS/400e series*. 29th Street Press, Loveland, CO, USA, 1997.

[SSF99] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. Eros: A fast capability system. In *Proc. 17th ACM Symp. OS Princip.*, pages 170–185, Charleston, SC, USA, December 1999.