

SECURITY

AN ADVANCED INTRODUCTION

COMP9242
2006/S2 Week 11

WHAT IS SECURITY?

Example 1: DOS:

- Single-user system with no access control
- Is it secure?
 - ... if it has no data?
 - ... if it contains the payroll database?
 - ... if it is on a machine in the foyer?
 - ... if it is in a locked room?
 - ... if it is behind a firewall?

WHAT IS SECURITY?

Example 1: Banking store's weekly earnings:

- Is it secure to
 - ask a random customer to do it?
 - ask many random customers to do it?
 - ask a staff member to do it?
 - ask several staff members to do it?
 - hire a security firm?
 - hire several security firms?
- Depends? On what?

SECURE SYSTEM

- Requires a *security policy*
 - specified *allowed* and *disallowed states* of a system
 - system needs to ensure that no disallowed state is ever entered
 - need OS *mechanisms* to prevent transitions from allowed to disallowed states
- Security policy needs to identify the *assets* to be secured
 - for computer security, the assets are typically data
- Perfect security is generally unachievable
 - need to be aware of *threads*
 - need to understand what *risks* can be tolerated

DATA SECURITY

Three aspects::

- Confidentiality: prevent theft of data
 - concealing data from unauthorised agents
 - *need-to-know principle*
- Integrity: prevent damage of data
 - trustworthiness of data: data correctness
 - trustworthiness of origin of data: authentication
- Availability: prevent denial of service
 - ensuring data is usable when needed

THREATS

- A *weakness* is a potential for a security violation
- An *attack* is an attempt by the *attacker* to violate security
 - generally implies exploiting a weakness
- A *threat* is a potential for an attack
- There is never a shortage of attackers, hence in practical terms
 - threat ⇒ attack
 - weakness ⇒ violation

THREATS

- Snooping
 - disclosure of data
 - attack on *confidentiality*
- Modification/Alteration
 - unauthorised change of data
 - attack on *data integrity*
- Masquerading/Spoofing
 - one entity impersonating another
 - attack on *authentication integrity*
 - delegation?
- Repudiation of origin
 - false denial of being source
 - attack on *integrity*
- Denial of receipt
 - false denial of receiving
 - attack on *availability* and *integrity*
- Delay
 - temporarily inhibiting service
 - attack on *availability*
- Denial of service
 - permanently inhibiting service
 - attack on *availability*

SECURITY POLICY

- Partitions system state into allowed and disallowed
- Ideally mathematical model
- In practice natural-language description
 - often imprecise, ambiguous, inconsistent, unenforceable
 - Example: transactions over \$10k require manager approval
 - but transferring \$10k into own account is no violation

SECURITY MECHANISMS

- Used to enforce security policy
 - computer access control (login authentication)
 - OS file access control system
 - controls implemented in tools
- Example:
 - Policy: only accountant can access finance system
 - Mechanism: on un-networked computer in locked room with only one key
- A secure system provides mechanisms that ensure
 - prevention
 - detection
 - recoveryof violations

ASSUMPTIONS

- Security is always based on assumptions
 - eg lock is secure, key holders are trustworthy
- Invalid assumptions *void* security!
- Problem: assumptions are often implicit and poorly understood
- Security assumptions must be
 - clearly identified
 - evaluated for validity

TRUST

- Systems always have *trusted entities*
 - hardware, operating system, sysadmin...
- Totality of trusted entities is the *trusted computing base* (TCB)
- Assumed to be *trustworthy*! Is it?

TRUSTED COMPUTING BASE

TCB: *The totality of protection mechanisms within a computer system — including hardware, firmware and software — the combination of which is responsible for enforcing a security policy.*

A TCB consists of one or more components that together enforce a unified security policy over a product or system.

The ability of the TCB to correctly enforce a security policy depends solely on the mechanisms within the TCB and on the correct input by system administrative personnel or parameters related to the security policy. (Gol99)

POTENTIALLY INVALID ASSUMPTIONS

- The security policy is unambiguous and consistent
- The mechanisms used to implement the policy are correctly designed
- The union of mechanisms implements the policy completely
- The mechanisms are correctly implemented
- The mechanisms are correctly installed and administered

ASSURANCE

- Process for *bolstering* (substantiating or specifying) trust
 - Specifications
 - unambiguous description of system behaviour
 - can be formal or informal
 - Design
 - justification that it meets specification
 - mathematical translation of specification or compelling argument
 - Implementation
 - justification that it is consistent with the design
 - mathematical proof or rigorous testing
 - by implication must also satisfy specification
 - Operation and Maintenance
 - justification that system is used as per assumptions in spec
- Assurance does not *guarantee* correctness/security!

ASSURANCE: ESTABLISHED APPROACHES

US DoD "Orange Book" (DoD86):

- Defines security classes
 - D: minimal protection
 - C1-2: discretionary access control
 - B1-3: mandatory access control
 - A1: verified design
- Systems can be certified to a certain class
 - very costly, hence only practicable for big companies
 - most systems only certified C2 (not more than Unix-style security)

ASSURANCE: ESTABLISHED APPROACHES

Common Criteria (NIS99):

- ISO standard, developed out of Orange Book
- Seven evaluation assurance levels (EALs)
 - EAL1: functionally tested
 - EAL2: structurally tested
 - EAL3: methodically tested and checked
 - EAL4: methodically designed, tested and reviewed
 - EAL5: semiformally designed and tested
 - EAL6: semiformally verified design and tested
 - EAL7: formally verified design and tested
- Higher levels imply more thorough evaluation
 - not necessarily better security
 - implementation *not* verified

SUMMARY

- Computer security is complex
 - depends on many aspects of computer system
- Policy defines security, mechanisms enforce security
- Important to consider:
 - what are the assumptions about threats and trustworthiness
 - incorrect assumptions ⇒ no security!
- Security is never absolute
 - given enough resources, mechanisms can be defeated
 - important to understand limitations
 - inherent tradeoff between security and usability
- Human factors are important
 - people make mistakes
 - people may not understand security impact of actions
 - people may be less trustworthy than thought

SECURITY POLICIES: 2 CATEGORIES

- Discretionary (user-controlled) policies
 - e.g. a_1 can read a_2 's object only with a_2 's permission
 - user decides about access
- Mandatory (system-controlled) policies
 - e.g. certain users cannot ever access certain objects
 - no user can change this

SECURITY POLICY MODELS

- Represent a whole class of security policies
- Most system-wide policies focus on confidentiality
 - e.g. military-style multi-level security models
 - classical example is Bell-LaPadula model (BL76)
 - most other based on this
- Other cases:
 - Chinese-wall policy focuses on conflict of interest
 - Clark-Wilson model focuses on separation of duty

BELL-LAPADULA MODEL

- Each object has a *security classification* $L(o)$
- Each agent has a *security clearance* $L(a)$
- Classifications and clearances from hierarchical *security levels*
 - eg: top secret > secret > confidential > unclassified
- Rule 1 ("no read up"):
 - a can read o only if $L(a) \geq L(o)$
 - standard confidentiality
- Rule 2 ("* Property"):
 - a can write o only if $L(a) \leq L(o)$
 - prevents *leakage* (accidental or by conspiracy)
 - problems:
 - logging
 - command chain

BELL-LAPADULA MODEL EXTENSIONS

- Can combine with discretionary access rights (read/write permissions on specific objects)
- Can add orthogonal *security categories* indicating types of data
 - restrict access to relevant categories
 - Denning's lattice model (Den76)

SECURITY MECHANISMS

- Used to implement security policies
- Based on *access control*
 - discretionary access control (DAC)
 - users can change access to objects
 - mandatory access control (MAC)
 - access rights centrally controlled

ACCESS RIGHTS

- Simple rights
 - read, write, execute/invoke, send, receive...
- Meta rights
 - copy (propagate a right to another agent)
 - own (change rights of an object or agent)

ACCESS CONTROL MATRIX

Agents	Objects			
	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

Note: Agents are objects too!

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

Represent 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 via *domain classes* (UNIX groups)
- Full ACLs used by Multics, Apollo Domain, Andrew FS, NTFS.
- Can have *negative rights*, to:
 - reduce "window of vulnerability",
 - simplify exclusion from groups.
- Sometimes implicit (process hierarchy).

ACCESS MATRIX IMPLEMENTATION: CAPABILITIES

Represent 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
 - Revocation: requires invalidation of capabilities from *all agents*
 - Amplification: special invocation capability.
 - Accessibility: requires inspection of all capability lists (how?)
 - Protection domain: explicit in capability list.
- Can have *negative rights*, to:
 - reduce "window of vulnerability",
 - simplify management of groups of capabilities.
- Successful commercial system: IBM System/38 *et fils*

CAPABILITIES

- Main advantage of capabilities is the fine-grained 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

- *Tag bit(s)* with every (group of) memory word(s):
 - Tags identify capabilities
 - Capabilities are used and copied like “normal” pointers
 - Hardware checks permissions on dereferencing capability
 - Modifications turn tags off
 - Only privileged instructions (kernel) can turn tags on
 - Propagation easy.
 - Restriction requires kernel to make new capability.
 - Revocation virtually impossible (memory scan!)
 - Amplification possible (see below).
 - Accessibility impossible to determine.
 - Protection domain difficult to establish.
- IBM System/38, AS/400, i-series; many historical systems.

PROTECTED PROCEDURE CALL (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 reconstituted.
 - Significant processing overhead with all I/O.

TAGGED CAPABILITIES SUMMARY

- Secure through hardware protection.
- Convenient for applications (appear as “normal” pointers).
- Checked by hardware ⇒ 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: system intervention to copy between clists.
 - Restriction: kernel to make new capability.
 - Revocation: kernel to remove cap from all (or specific) clists.
 - Accessibility can only be determined by scanning all clists.
 - Protection domain is explicitly represented in clist.
- Hydra (CJ75), Mach (RTY+88), KeyKOS (BFF+92), 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 ⇒ apps use “normal” pointers.
- Fast validation (clist check is simple, validation cached by MMU).
- Propagation requires marshaling and kernel intervention.
- Reference counting possible to detect unaccessible objects.

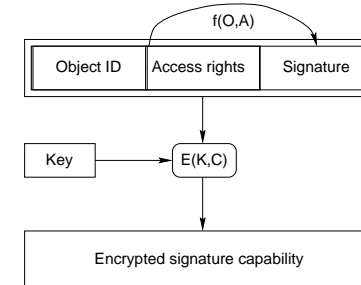
SPARSE CAPABILITIES

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.

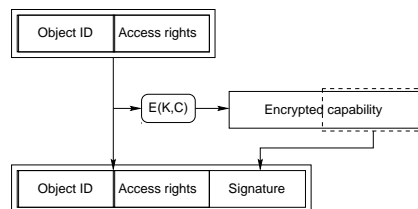


- + tamper proof via encryption with secret kernel key
- + can freely be passed around
- need to decrypt on each validation
- users do not know which object capability refers to

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

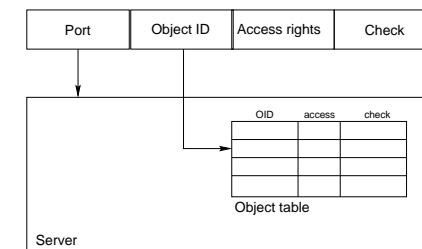
EXAMPLE: SIGNATURE CAPABILITIES

"Second Migration Scheme" (GL79)



Object ID visible, yet still tamper proof.

EXAMPLE: AMOEBAS CAPABILITIES

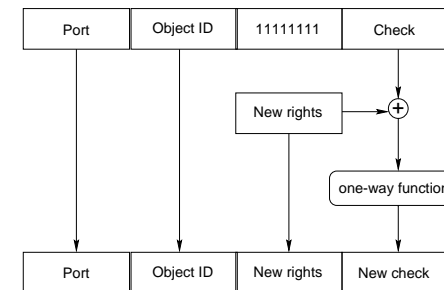


Appropriate for user-level servers (MT86).

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.

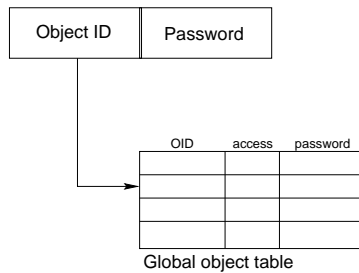
IMPROVED VERSION (NOT IMPLEMENTED)

- 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



- 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.
 - \rightarrow Validation done by kernel on access or presentation and cached by MMU.
 - \rightarrow Propagation easy, as capabilities are "normal" data.
 - \rightarrow Restriction requires kernel to make new capability.
 - \rightarrow Revocation done by kernel removing entry from object table.
 - \rightarrow Amplification possible similar to AS/400.
 - \rightarrow Accessibility is impossible to determine.
 - \rightarrow Protection domain is known to kernel.

SPARSE CAPABILITIES SUMMARY

- Statistically secure (like encryption).
- Validation at mapping time \Rightarrow 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.

CONFINEMENT

- Problem 1: Executing untrusted code
 - \rightarrow you downloaded a game from the internet
 - \rightarrow how can you be sure it doesn't steal all your data?
- Problem 2: Digital rights management (DRM)
 - \rightarrow you own copyrighted material (e.g. movie)
 - \rightarrow you want to let others view it (for a fee)
 - \rightarrow how can you be sure the clients don't make unauthorised copies?
- Need to *confine* program (game, viewer) so it cannot leak data
- Cannot be done with most protection systems!
 - \rightarrow not with UNIX or most other ACL-based systems
 - \rightarrow not with most tagged or sparse capability systems
 - \rightarrow multi-level security has some inherent confinement but wouldn't help for DRM

CONFINEMENT

- Some protection models can confine in principle
 - e.g. segregated caps system requires all caps to be presented explicitly
 - can instruct system not to accept any more caps from confined process
 - EROS has a formal proof of its confinement properties (SW00)
- In practice difficult to achieve confinement due to *covert channels*

COVERT CHANNELS

- Information flow that is not controlled by a security mechanism
 - Security requires *absence of covert channels*
- Two types of covert channels
 - Covert storage channel uses an attribute of a shared resource
 - typically some meta-data, like existence or accessibility of an object
 - global names create covert storage channels
 - in principle subject to access control
 - a sound access-control system should be *free* of covert channels
 - Covert timing channel uses a temporal ordering relationship among accesses to a shared resource
 - outside access control system
 - difficult to reason about
 - difficult to prevent

COVERT TIMING CHANNELS

- Can be created via a shared resource whose behaviour can be monitored
 - network bandwidth
 - CPU load
 - response time
 - locks
- Require access to a time source
 - real-time clock
 - anything else that allows unrelated processes to synchronise
- Critical issue is bandwidth
 - in practice the damage is limited if the bandwidth is low
 - e.g. DRM doesn't care about low-bandwidth channels
 - beware of amplification (e.g. leaking a password capability)!

DESIGN PRINCIPLES FOR SECURE SYSTEMS

- Least privilege
- Fail-safe defaults
- Economy of mechanism
- Complete mediation
- Open design
- Separation of privilege
- Least common mechanisms
- Psychological acceptability

LEAST PRIVILEGE

- Agent should only be given the minimal rights needed for task
 - minimal protection domain (PD)
 - PD determined by *function*, not *identity*
 - Unix `root` is bad
 - *role-based access control* (RBAC) tries to address this
 - rights added as needed, removed when no longer needed

FAIL-SAFE DEFAULTS

- Default action is no access
 - if action fails, system remains secure
 - if security administrator forgets to add rule, system remains secure
 - “better safe than sorry”

ECONOMY OF MECHANISM

- KISS principle of engineering
- Less code/features/stuff ⇒ less to get wrong!
 - makes it easier to fix if something does go wrong
 - complexity is the natural enemy of security
- Also applies to interfaces, interactions, protocols...
- Minimal trusted computing base!

COMPLETE MEDIATION

- Check *every* access
 - Violated in Unix file access:
 - access rights checked at `open()`
 - access then enabled as long as file is open
 - ... even if access rights change
 - Also implies that any rights propagation must be controlled
 - issue for tagged or sparse capability systems
- In practice, this is in conflict with performance
 - caching of buffers, file descriptors etc
 - unacceptable performance in distributed systems
- Should at least limit window of opportunity
 - e.g. guarantee caches to be flushed after some fixed period

OPEN DESIGN

- Security must *not* depend on secrecy of design or implementation
 - the TCB must be open to scrutiny
 - *security by obscurity* is poor security
 - e.g. the US government's Clipper initiative ('92)
- Note that this does not rule out passwords or secret keys
 - but the way they are created/used requires careful *cryptoanalysis*

SEPARATION OF PRIVILEGE

- Require combination of conditions to grant privilege
 - e.g. user is in group `wheel` *and* knows the root password
 - closely related to least privilege

LEAST COMMON MECHANISMS

- Avoid sharing mechanisms
 - shared mechanism ⇒ shared channel
 - potential covert channel
- Inherent conflict with other design imperatives
 - simplicity ⇒ shared mechanisms

PSYCHOLOGICAL ACCEPTABILITY

- Security mechanisms should not add to difficulty of use
 - hide complexity introduced by security mechanisms
 - ensure ease of installation, configuration, use
 - systems are used by *humans*
- Inherently problematic
 - security inherently inhibits ease of use
 - idea is to minimise impact
- Security-usability tradeoff is to a degree unavoidable

REFERENCES

- (APW86) Mark 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 *7th Symp. Comp. Arch.*, pages 245–250. ACM/IEEE, May 1980.
- (BFF+92) Alan C. Bromberger, A. Peri Frantz, William S. Frantz, Ann C. Hardy, Norman Hardy, Charles R. Landau, and Jonathan S. Shapiro. The KeyKOS nanokernel architecture. In *USENIX WS Microkernels & other Kernel Arch.*, pages 95–112, Seattle, WA, USA, Apr 1992.
- (BL76) D.E. Bell and L.J. LaPadula. Secure computer system: Unified exposition and Multics interpretation. Technical Report MTR-2997, MITRE Corp., Mar 1976.
- (CJ75) E. Cohen and D. Jefferson. Protection in the HYDRA

REFERENCES

- operating system. In *5th SOSp*, 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. *Trans. Comp. Syst.*, 12:271–307, 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.
- (Den76) Dorothy E. Denning. A lattice model of secure information flow. *CACM*, 19:236–242, 1976.
- (DoD86) US Department of Defence. *Trusted Computer System Evaluation Criteria*, 1986. DoD 5200.28-STD.
- (GL79) V.D. Gilgor and B.G. Lindsay. Object migration and

REFERENCES

- authentication. *Trans. Softw. Engin.*, 5:607–611, 1979.
- (Gol99) Dieter Gollmann. *Computer Security*. Wiley, 1999.
- (HEV+98) Gernot Heiser, Kevin Elphinstone, Jerry Vochtelloo, Stephen Russell, and Jochen Liedtke. The Mungi single-address-space operating system. *Softw.: Pract. & Exp.*, 28(9):901–928, Jul 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.
- (NIS99) US National Institute of Standards, <http://csrc.nist.gov/cc/>. *Common Criteria for IT Security Evaluation*, 1999. ISO Standard 15408.
- (RTY+88) Richard Rashid, Avadis Tevanian, Jr., Michael Young, David Golub, Robert Baron, David Black, William J.

REFERENCES

- Bolosky, and Jonathan Chew. Machine-independent virtual memory management for paged uniprocessor and multiprocessor architectures. *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 *17th SOSp*, pages 170–185, Charleston, SC, USA, Dec 1999.
- (SW00) Jonathan S. Shapiro and Samuel Weber. Verifying the EROS confinement mechanism. In *Symp. Security & Privacy*, 2000.