TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science, Institute of Systems Architecture, Operating Systems Group

Distributed Operating Systems:

Security: Foundations, Security Policies, Capabilities

2011

Marcus Völp / Hermann Härtig

Can you trust your system?

- to protect your privacy / credentials / valuable data?
- to grant only trusted programs access to your data?
- to grant access to your data when and only when a trusted program needs it?

Can you trust your system?

- to protect your privacy / credentials / valuable data?
- to grant only trusted programs access to your data?
- to grant access to your data when and only when a trusted program needs it?

How can you trust your system?

- Assurance because
  - trust the developer / company (or you sue them)
  - quality assuring processes (e.g., independent test team)
  - certification (e.g., ISO 9000; Common Criteria; DO 178b)
  - formal verification (i.e., CC EAL 7+, (old) BSI GISA, ...)

Formal Verification

- Abstract Mathematical Model
  - describes your system in a way that is
    - precise,
    - unambiguous,
    - “small” (to be understood in its entirety)
    - “understandable”

- Formal Specification
  - describes the behavior / properties

- “Correctness” Proof
  - connects abstract model and specification / properties

- Refinement Proofs
  - connects the abstract model with the implementation

Formal Verification

- Abstract Mathematical Model
  - describes your system in a way that is
    - precise,
    - unambiguous,
    - “small” (to be understood in its entirety)
    - “understandable”

- Formal Specification
  - describes the behavior / properties

- “Correctness” Proof
  - connects abstract model and specification / properties

- Refinement Proofs
  - connects the abstract model with the implementation

**11 PY to verify seL4,
a 10KLOC microkernel**

Security Evaluations

- Common Criteria (EAL 7)
  - Formal top level specification
  - Informal (through tests) correspondence of source code to abstract specification

- GISA IT Security Evaluation Criteria (Q7)
  (an old proposal for CC-EAL 7 from 1989)
  - “The machine language of the processor used shall to a great extent be formally defined.”
  - “The consistency between the lowest specification level and the source code shall be formally verified.”
  - “The source code will be examined for the existence of covert channels, applying formal methods. It will be checked that all covert channels detected which cannot be eliminated are documented. [...]”

TECHNISCHE UNIVERSITÄT DRESDEN

Outline

- Introduction
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model
- (Covert Channels and
  Flow Sensitive Security Type Systems)

Security Policies

- **Example:**
  - “Only the owner of a file and root shall have the permission to write this file.“
- **Security Policy**
  - Defines what is allowed / secure and what is not allowed / insecure
- **Secure System**
  - System that enforces a security policy

Proving Security

- **1st Ingredient**
  - abstract model of static behavior: state

$\Sigma := \{ (U_{life}, F_{life}, owner, rights, u_{current}) \}$ with

$U_{life} \subseteq Users$	set of “life / existing” users
$F_{life} \subseteq Files$	set of “life / existing” files
$u_{current} \in U_{life}$	the current user
owner: $F_{life} \rightarrow U_{life}$	who possesses which file
rights: $U_{life} \times F_{life} \rightarrow \wp(R)$	permissions

Example:

$\sigma \in \Sigma :=$ ({root, marcus, hermann},
{foo.txt, bar.txt}, root,

{(foo.txt, marcus), (bar.txt, hermann)},
{(root, foo.txt, {rw}), (hermann, bar.txt, {w}) })

Proving Security

- **2nd Ingredient**
  - abstract model of dynamic behavior: state transitions

C := { read (file), write(file), create(user), delete(file)
chmod(user, file, rights), ...}

Examples:

$\sigma \xrightarrow{\text{read(bar.txt)}} \sigma$

$\sigma \xrightarrow{\text{delete(bar.txt)}} \sigma'$ with

σ' := ({root, marcus, hermann},
{foo.txt, ~~bar.txt~~}, root,

{(foo.txt, marcus), ~~(bar.txt, hermann~~)},
{(root, foo.txt, {rw}), (~~hermann, bar.txt, {w}~~) })

Proving Security

- **2^{nd} Ingredient**
  - abstract model of dynamic behavior: state transitions

C := { read (file), write(file), create(user), delete(file)
chmod(user, file, rights), ...}

Examples:

$\sigma \xrightarrow{\text{read(bar.txt)}} \sigma$

$\sigma \xrightarrow{\text{delete(bar.txt)}} \sigma'$ with

if $u_{current}$ = root v owner(bar.txt, $u_{current}$) **then**
σ' := ({root, marcus, hermann},
{foo.txt, ~~bar.txt~~}, root,

{(foo.txt, marcus), ~~(bar.txt, hermann)~~},
{(root, foo.txt, {rw}), (~~hermann, bar.txt, {w}~~) })
else
σ' := σ
endif

Proving Security

- **3rd Ingredient**
  - property

$P(\sigma) := \forall\ u,f.\ w \in rights(u,f) => owner(f,u)\ v\ u = root$

but

$\sigma := (\{marcus, hermann\}, \{bar.txt\}, marcus,$
$\{(bar.txt, hermann)\}, \{(marcus, bar.txt, \{rw\})\}) \in \Sigma$

Proving Security

- **3rd Ingredient**
  - property

    $P(\sigma) := \forall\, u,f.\ w \in \text{rights}(u,f) \Rightarrow \text{owner}(f,u) \vee u = \text{root}$

    where $\sigma \in \Sigma_{\text{reachable}}$

  - initial state σ_0
  - $\Sigma_{\text{reachable}} := \{\ \sigma \in \Sigma \mid \exists\, c_0, c_1, \ldots\ \sigma_0 \xrightarrow{c_0,\, c_1,\, \ldots} \sigma\ \}$

Proving Security

- **4th Ingredient**
  - correctness / security proof:

    $P(\sigma)$ is an invariant for all reachable states

  - often by induction over all command sequences C^*

    C := { create(user), delete(file), chmod(user, file, rights) }

    ~> $\neg$ $P(\sigma)$ because chmod(hermann, foo.txt, {w})

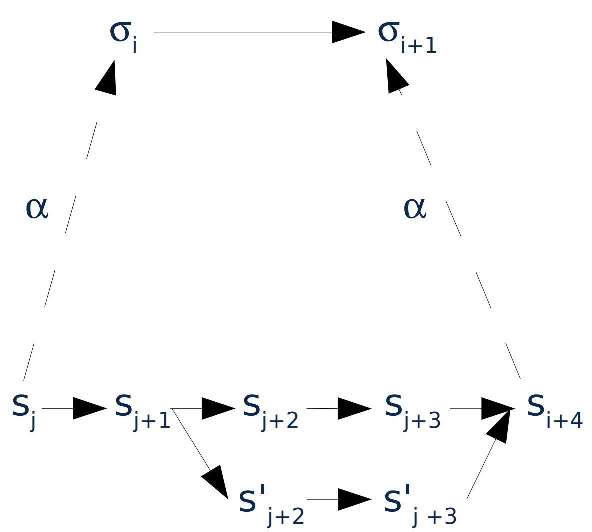
    but e.g.,

    chmod'(u, f, R)(σ) :=
    if u = root v owner(f, u) **then**
    chmod(u, f, R)(σ)
    else
    σ
    endif

Proving Security

- **5th Ingredient**
  - refinement proof:

```
chmod(u, f, R)(σ) :=
  if u = root v owner(f, u) then
        σ with rights (u, f) := R
  else
        σ
  endif
```




```
sys_chmod:
  parse_parameters();
  owner = file.owner;
  if (current_thread->user == root ||
      current_thread->user == owner)
  {
        file->set_acl(user, rights);
  }
```

Security Policies - Definition

[Bishop: Computer Security Art and Science]

- **Security Policy**

  A *security policy* P is a statement that partitions the states Σ of a system into a set of authorized (or secure) states

  (e.g., $\Sigma_{sec} := \{ \sigma \in \Sigma \mid \sigma \in \Sigma_{reachable} \wedge P(\sigma) \}$)

  and a set of unauthorized (or non-secure) states.

- **Secure System**

  A secure system is a system that starts in an authorized state and that cannot enter an un-authorized state.

  i.e., $\Sigma_{reachable} \subseteq \Sigma_{sec}$

Confidentiality, Integrity, Availability

- **Confidentiality**

  prevent unauthorized disclosure of sensitive information (information leakage).

- Definition:

  Information or data I is *confidential* with respect to a set of entities X if no member of X can obtain information about I.

  Example: the PIN of my EC-Card is XXXX

Confidentiality, Integrity, Availability

- **Integrity**

  correctness of information or data

- Definition 1:

  Information I is *integer* if it is current, correct and complete

Confidentiality, Integrity, Availability

- **Integrity**

  correctness of information or data

- Definition 1:

  Information I is *integer* if it is current, correct and complete

- Definition 2: (crypto)

  Either information is current, correct, and complete (Def 1) or it is possible to **detect** that these properties do not hold

Confidentiality, Integrity, Availability

- **Integrity**

  correctness of information or data

- Definition 1:

  Information I is *integer* if it is current, correct and complete

- Definition 2: (crypto)

  Either information is current, correct, and complete (Def 1) or it is possible to **detect** that these properties do not hold

- **Recoverability**

  Eventually, damaged information can be recovered

Confidentiality, Integrity, Availability

- **Availability**

  accessibility of information, services and data

- <u>Definition:</u>

  A resource I is available with respect to X if all members of X can access I.

- in practice, availability has also quantitative aspects:
  - real-time systems:

    I is available within t milliseconds

  - reliability:

    the probability that I is **not** available is less than 10^{-6}

Security Policies - Classification

- **Concern**

  - confidentiality — e.g., Bell La Padula (Document Mgmt)
  - integrity — e.g., Biba (Inventory System)
  - availability
  - hybrid — e.g., Chinese Wall (Clinical Information)

- **Level of Enforcement**

  - **discretionary**

    A user can allow or deny access to its objects

  - **mandatory**

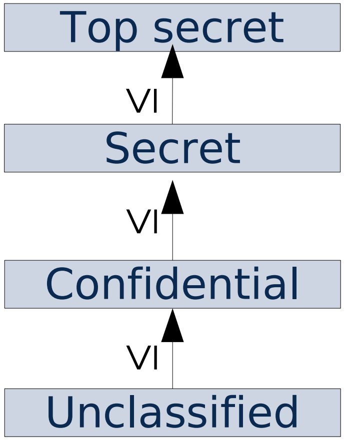
    System-wide rules control who may access an object

Bell-LaPadula Policy '73 (simple version)

- **Concern:** confidentiality

set of secrecy levels: L

higher secrecy level indicates more sensitive information; greater need to keep this information confidential



total order: ≤

domain:

- each subject has a *security clearance:* dom(s) ∈ L
- each object has a *security classification:* dom(o) ∈ L

Bell-LaPadula Policy '73 (simple version)

- **Policy: (L, ≤, dom)**

  rules for reading / writing

  - **simple security condition**

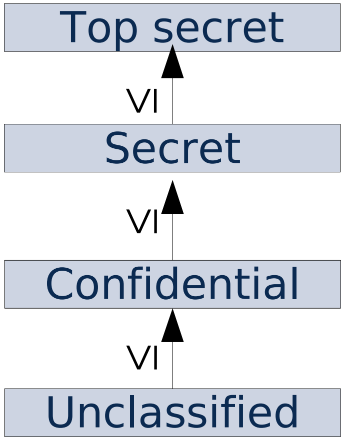
    a subject s can read only lower or equally classified objects o

    i.e., s can read o <=> dom(o) ≤ dom(s)

  - *** - property**

    a subject s can write only higher or equally classified objects o

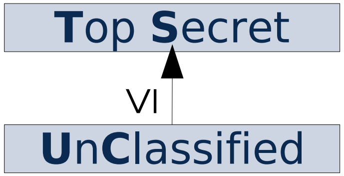
    i.e., s can write o <=> dom(s) ≤ dom(o)



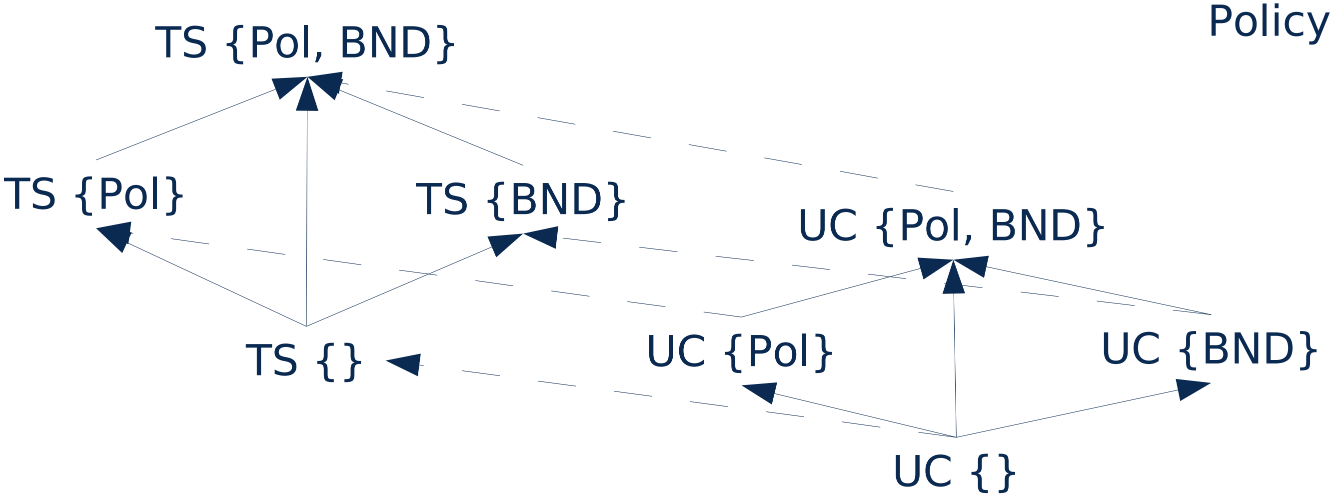
Bell-LaPadula: Multi-Level Security Policy

- **Policy: (L, ≤, dom)**

  ≤ is a partial order
  (L, ≤) form a lattice



Categories:
Policy, BND



Bundesverfassungsschutzgesetz §17 - §26:

in general, no information exchange between the BND and the Police

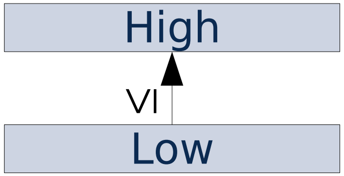
Biba '77

- **Concern:** Integrity (prevent damage)

  (L, ≤, dom) dual to MLS

  high integrity information must not be tainted with low integrity data.

  - s can read o <=> dom(s) ≤ dom(o)
  - s can write o <=> dom(o) ≤ dom(s)



Biba '77 Low Water Mark

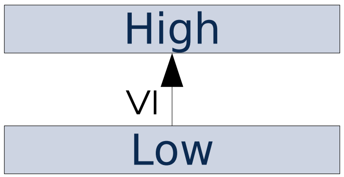
- **Concern:** Integrity (prevent damage)

  (L, ≤, dom) dual to MLS

  high integrity information must not be tainted with low integrity data.

  - ~~s can read o <=> dom(s) ≤ dom(o)~~
  - if s reads o then dom'(s) = min(dom(s), dom(o))

  - s can write o <=> dom(o) ≤ dom(s)

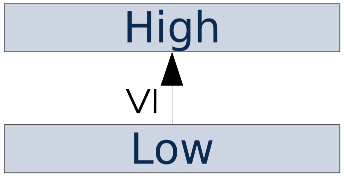


Biba '77 Low Water Mark

- **Concern:** Integrity (prevent damage)

  (L, ≤, dom) dual to MLS

  high integrity information must not be tainted with low integrity data.

  - ~~s can read o <=> dom(s) ≤ dom(o)~~
  - if s reads o then dom'(s) = min(dom(s), dom(o))

  - s can write o <=> dom(o) ≤ dom(s)

- **Problem:** label creep

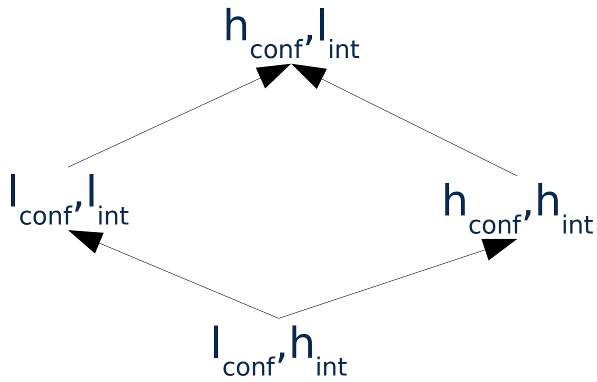
  subject clearances decrease over time
  no means to “clean” a tainted subject

Denning '76 + Sandhu '93

- Confidentiality and Integrity are dual and can be represented in the same lattice:

Confidentiality: $l_{conf} \leq h_{conf}$

Integrity: $h_{int} \leq l_{int}$



Brewer '89: Chinese Wall

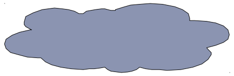
- **Concern:** Conflict of Interest (Integrity + Confidentiality)

  Example: British stock exchange
  a trader must not represent two competitors

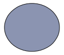
  Company Datasets (CD):
  set of objects (files) related to a company

  Conflict of Interest Class (COI):
  CDs of companies in competition

  Sanitized Objects:
  cleared to the public



  Subjects (e.g., the trader)



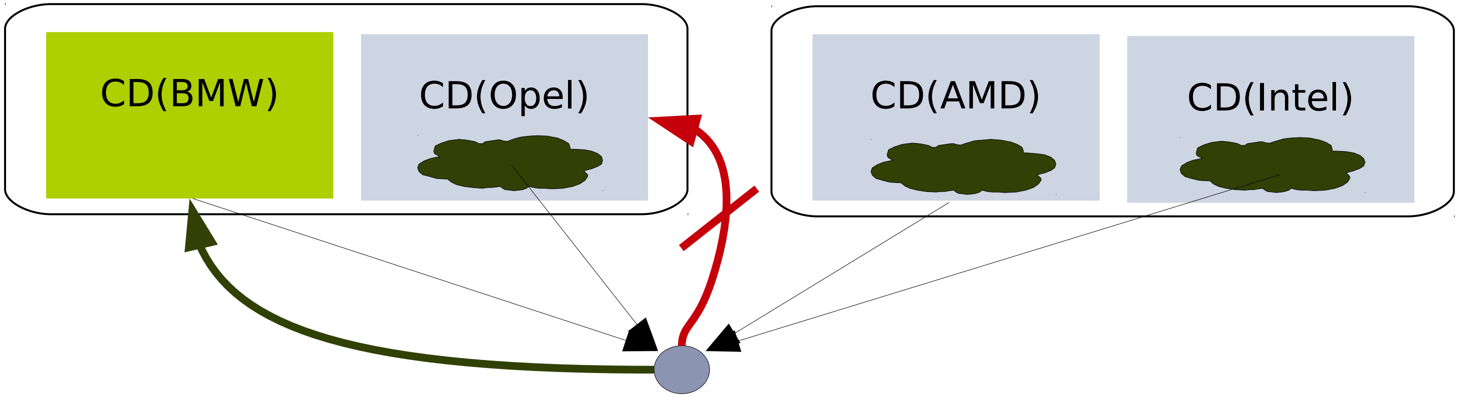
Brewer '89: Chinese Wall

- *** Property**

  s can write o <=>

  s can read o
  and
  if s can read an unsanitized object o' then o' must belong to the same company as o

  i.e., ∀ o'. s can read o' => CD(o') = CD(o)



Outline

- Introduction
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model
- (Covert Channels and
  Flow Sensitive Security Type Systems)

Access Control Matrix

Subjects S
Objects O
Entities E = S ∪ O
Rights R

Matrix: S x E x R

	o_1	o_2	s_1	s_2
s_1	r,w	r	r,w	r
s_2	r,w	-	w	r,w

Operations:

- read / write entity
- create subject / object
- destroy subject / object
- **enter / delete R into cell (s,o)**

Access Control List

Subjects S
Objects O
Entities $E = S \cup O$
Rights R

List per Entity: S x R

	o_1	o_2	s_1	s_2
s_1	r,w	r	r,w	r
s_2	r,w	-	w	r,w

Abbreviations:

- owner / group e.g., Unix [user; group; all]
- wildcards e.g., sysadmin_*

Conflicts:

- e.g., u – r; g + r resolved by order of occurrence / rules

Capabilities

Subjects S
Objects O
Entities $E = S \cup O$
Rights R

"List" per Subject: E x R

	o_1	o_2	s_1	s_2
s_1	r,w	r	r,w	r
s_2	r,w	-	w	r,w

more in a minute

Principle of Attenuation

German: Abschwächung / Verminderung

A subject s must not be able to give away rights that it does not possess

	o_1	o_2	s_1	s_2
s_1	r,w	r	r,w	r
s_2	r,w	-	w	r,w

Problem: ACMs cannot enforce the principle of attenuation

- e.g., s_1.enter w into (s_2, o_2)

Solution:

replace enter r into (s,o) with:

s'.grant R into (s,o) :=
if R $\subseteq$ (s',o) **then** enter R into (s,o)

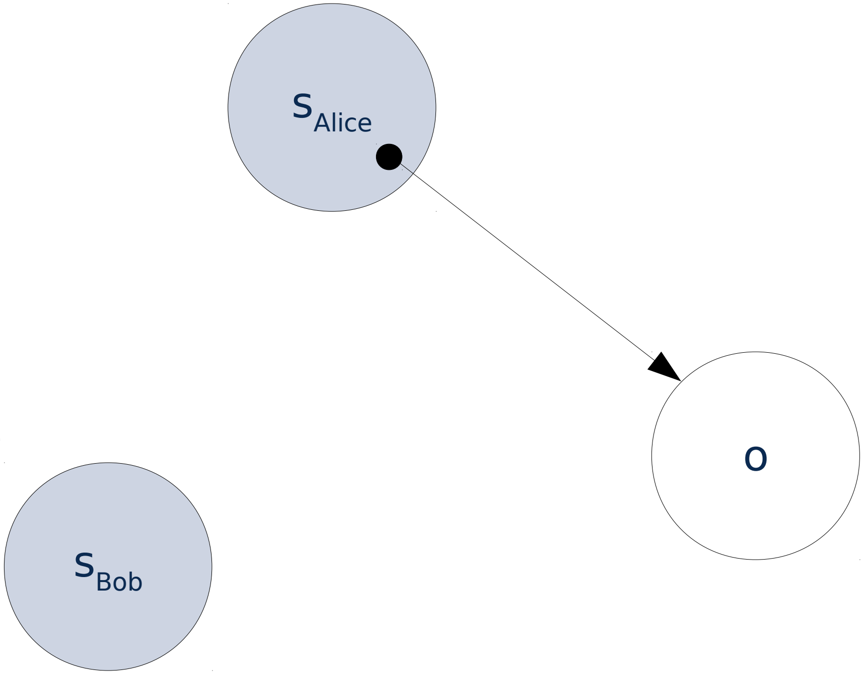
Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / grant
  - diminish / remove



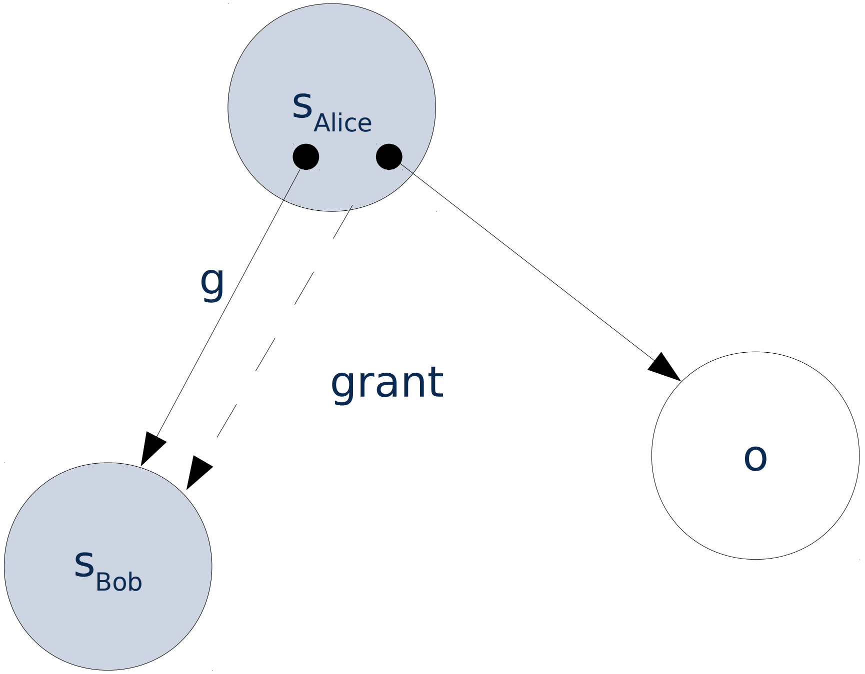
Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / **grant**
  - diminish / remove



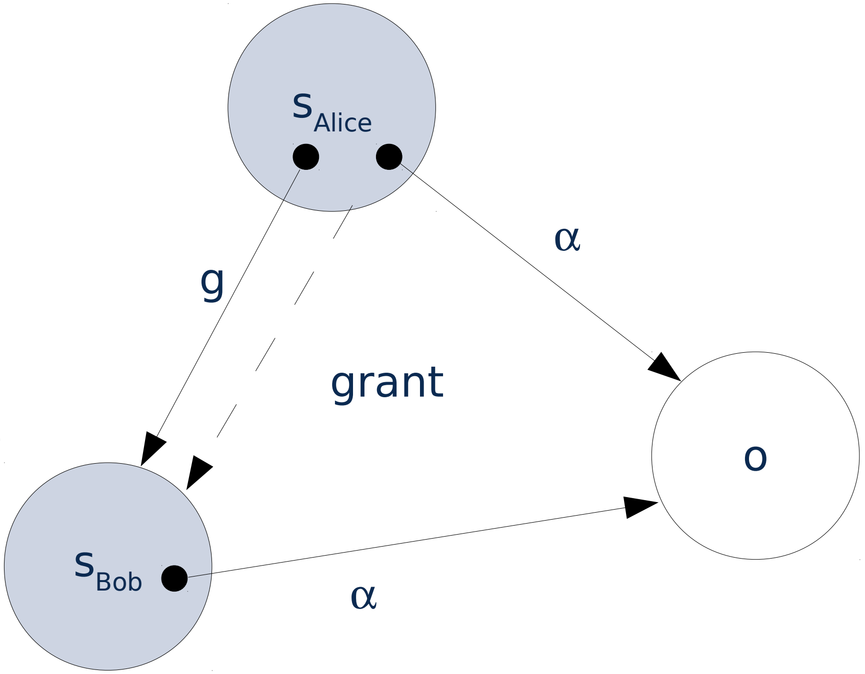
Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / **grant**
  - diminish / remove



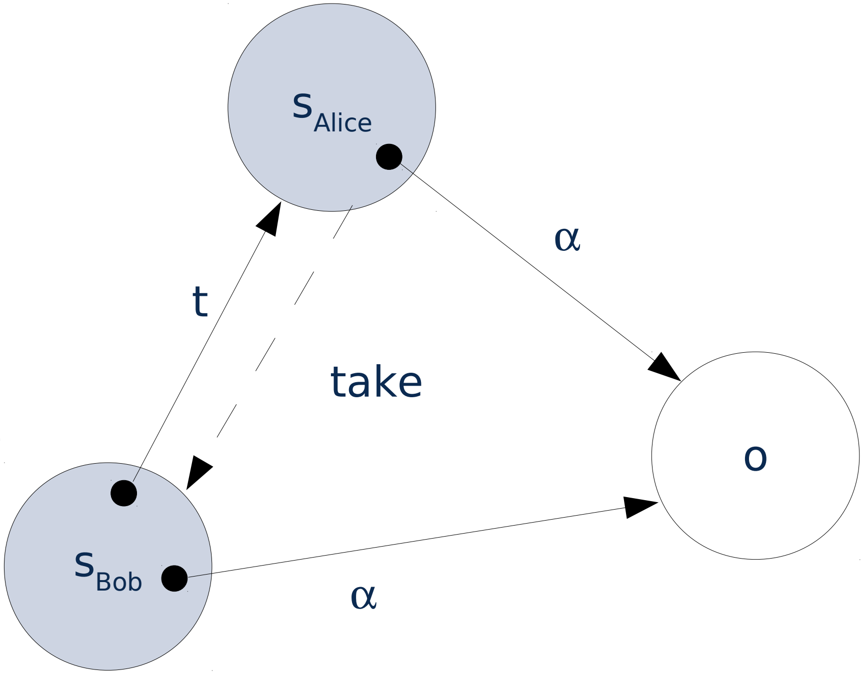
Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - **take** / grant
  - diminish / remove



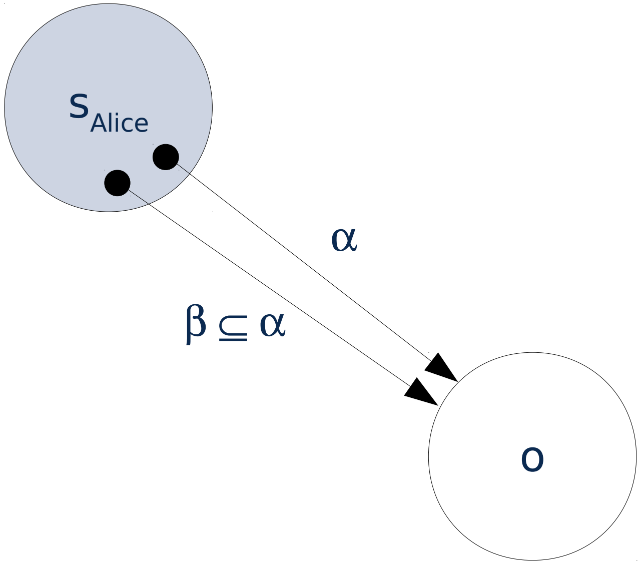
Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / grant
  - **diminish** / remove



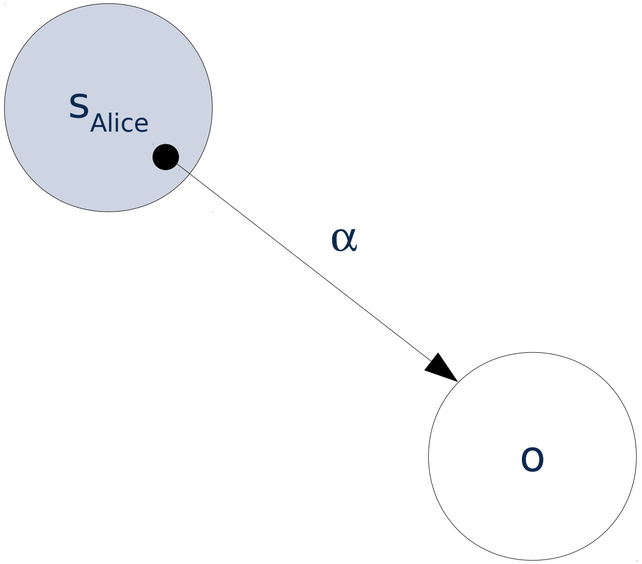
Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / grant
  - diminish / **remove**



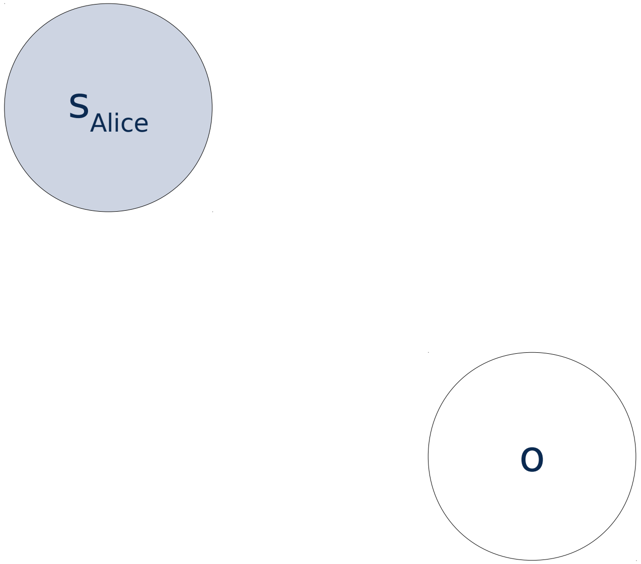
Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / grant
  - diminish / **remove**



Capabilities

Implementation:

Software:	OS protected segment / memory page
Hardware:	Cambridge CAP / TLB
Cryptography:	Amoeba

Problems:

- How to control the propagation of capabilities?
- How to revoke capabilities?

Propagation of Capabilities

Problem is dual to controlling ACM / ACL modifications

Permissions on channel capabilities:
take permission (t); grant permission (g)

Permission on the capability:
copy permission

Right-diminishing channels:
extension to the take-grant model by J. Shapiro

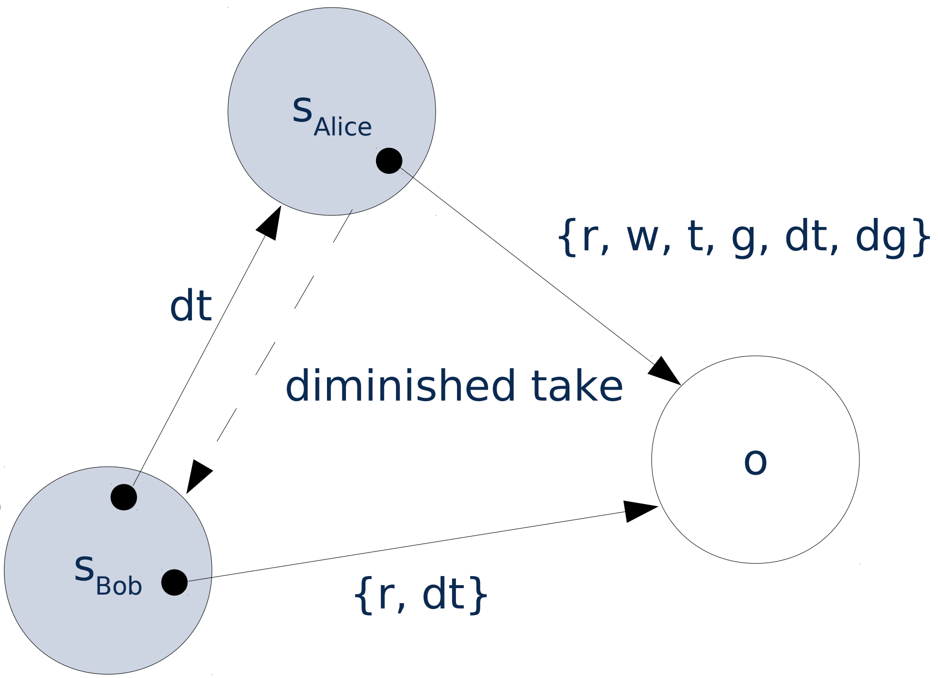
Propagation of Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / grant
  - diminish / remove
  - **diminished take**
  - diminished grant

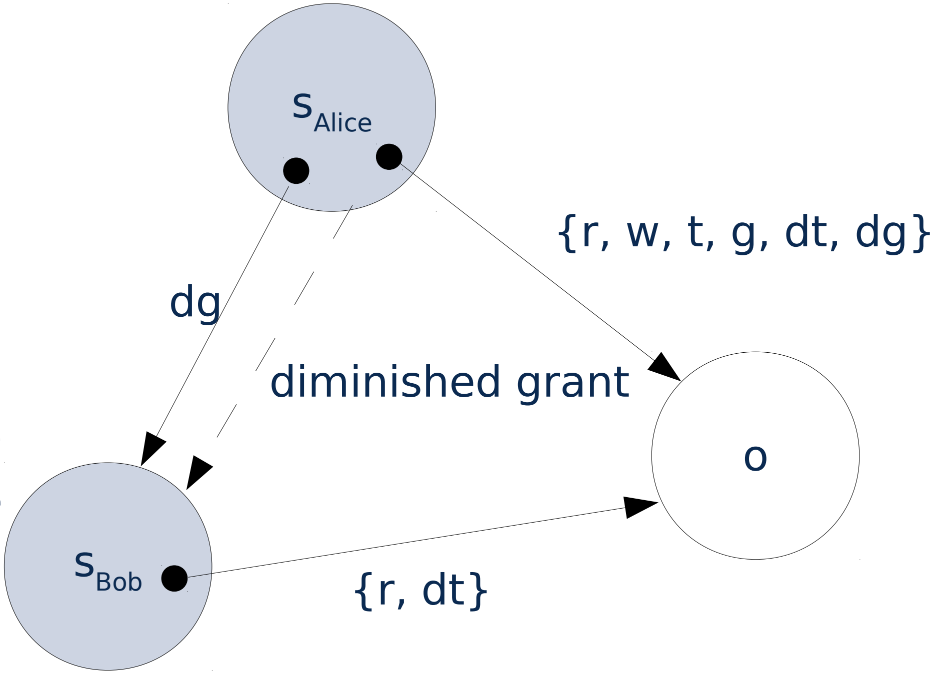


Propagation of Capabilities

unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

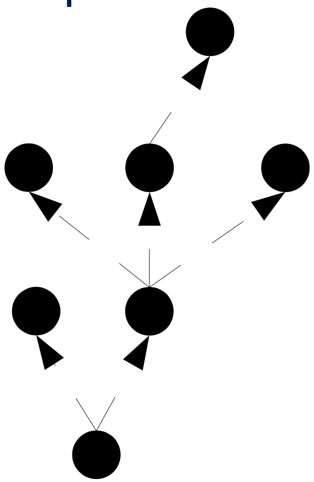
Operations:
- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / grant
  - diminish / remove
  - **diminished take**
  - diminished grant



Capability Revocation

Amoeba: leases – invalid after a certain amount of time

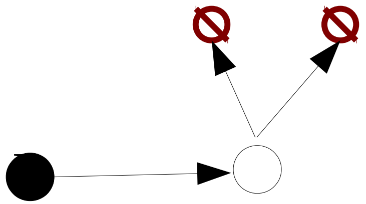
L4: find and invalidate all direct and indirect copies



Eros: indirection objects

use stored capabilities
but no take / grant

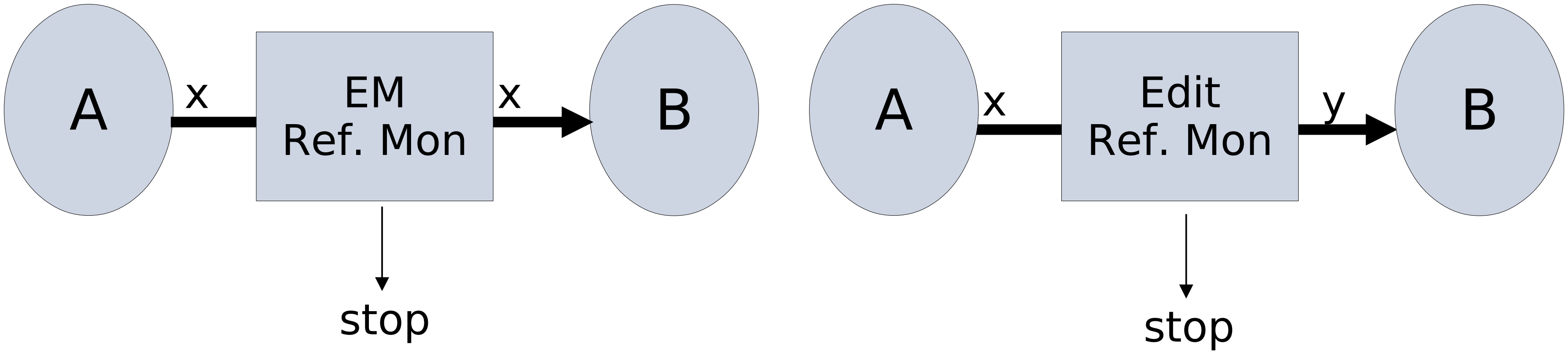
revoke by destruction



Reference Monitors

EM: suppress or pass

Edit: modify message

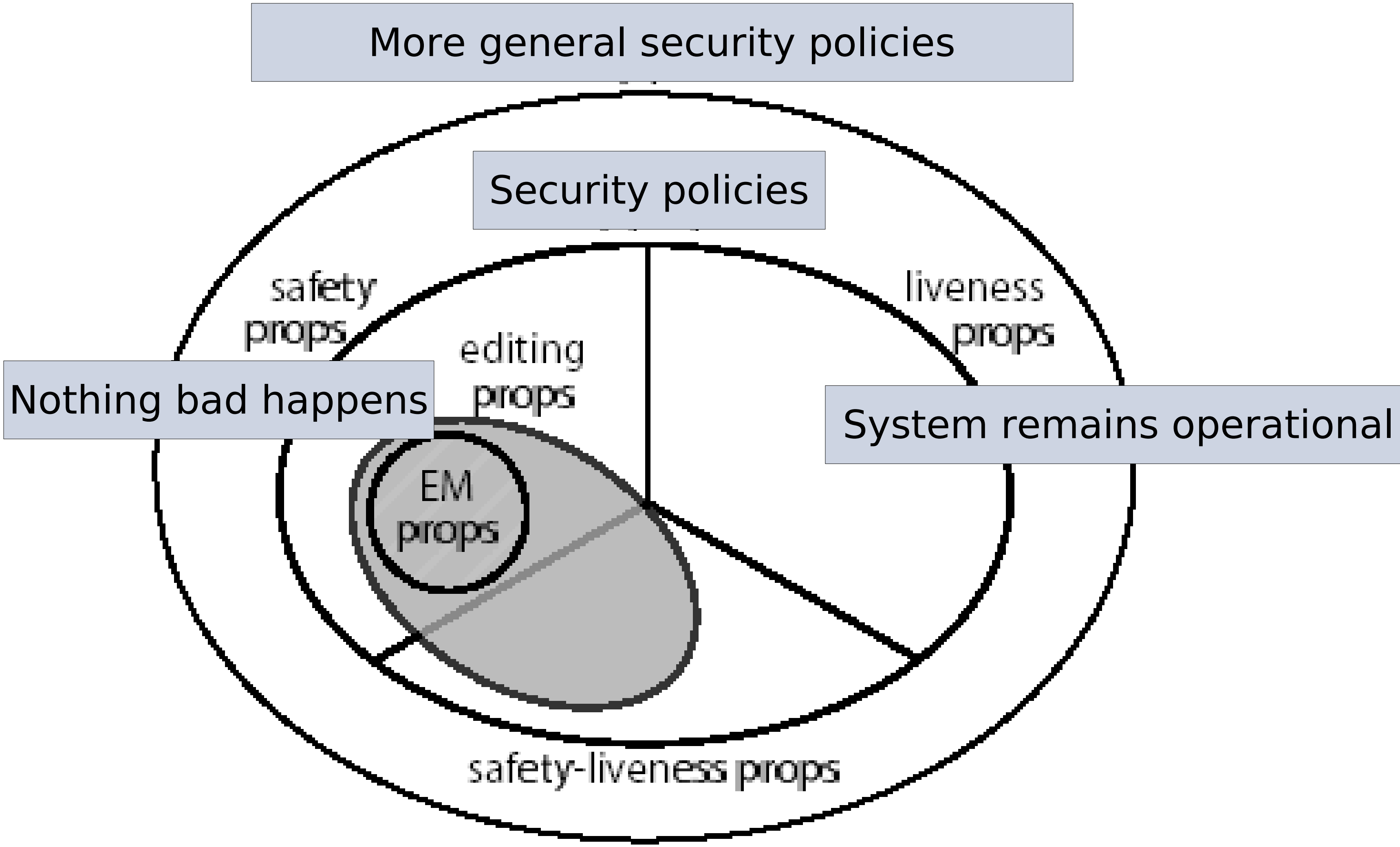


Schneider '98 / Bauer '02:

Theoretical results on the set of security policies that are enforceable with EM / Edit automata

!!! Results are in part based on a different system model !!!

Reference Monitors



Outline

- Introduction
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model
- (Covert Channels and
  Flow Sensitive Security Type Systems)

Leakage

given a system S and a security policy P:

decide whether S can enter a state in which s can access o with right r (i.e., whether r is leaked into (s,o))

Theorem:

For a system S with a generic ACM it is in general undecidable whether S leaks r into (s, o).

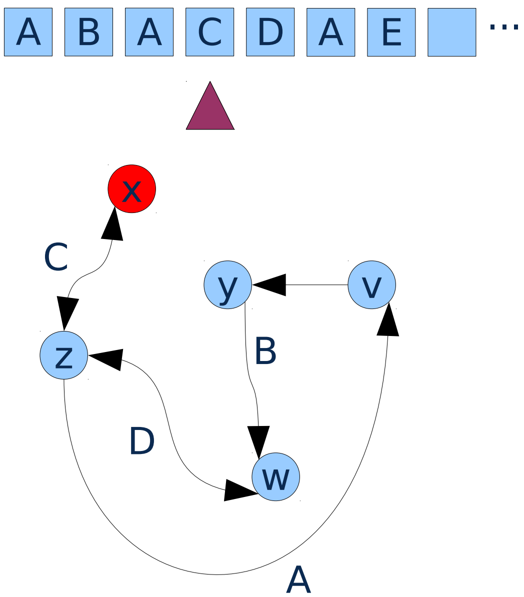
Proof:

By reduction to the halting problem

Turing Machine

infinite tape

tape symbols M: A, B, C, ...
state automaton K: x, y, z, ...
head



Operations:

- read symbol at head
- perform a transition step of the automaton based on this symbol
- write a new symbol to the tape
- move head one step to the left or to the right

$$\delta: K \times M \rightarrow K \times M \times \{L, R\}$$

Halting Problem

Given a turing machine TM and a program P, find a program of the TM that decides whether P will terminate (halt)

TM $\cong$ universal TM $\cong$ while

Theorem: the halting problem is undecidable

Halting Problem

TM ≅ universal TM ≅ while

Theorem: the halting problem is undecidable

Proof: by contradiction

assume such a program P exists; write two programs:

does_P_terminate_on_input_E (P, E) :=
if P(E) terminates { return true} else {return false }

test (P) := while (does_P_terminate_on_input_E(P, P))

now, if does_P_terminate_on_input_E(test, test) returns true, test(test) must terminate [*if condition*]

but then the condition of the while loop is true, which means test(test) will not terminate 

=> there cannot be a program that decides for all P, E whether P terminates on E

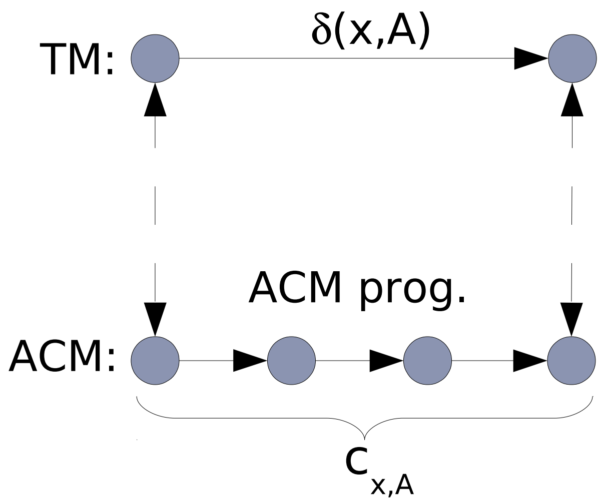
Leakage is Undecidable

Proof: by reduction to the halting problem

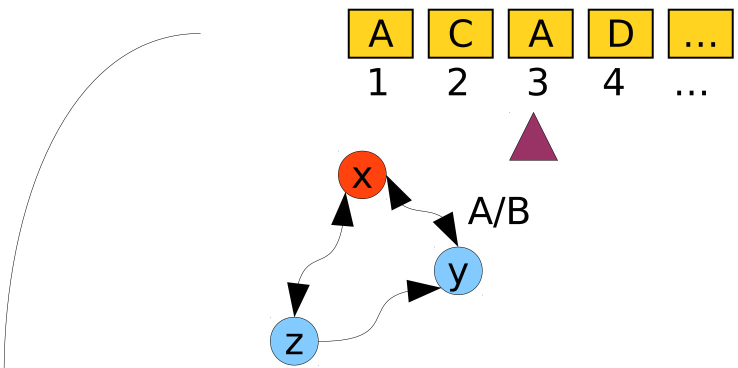
1. Simulate a TM with the ACM
2. Define a correspondence relation such that
   r is leaked to (s,o) <=> TM halts

=> leakage in the ACM could be used to solve the halting problem, which is known to be undecidable

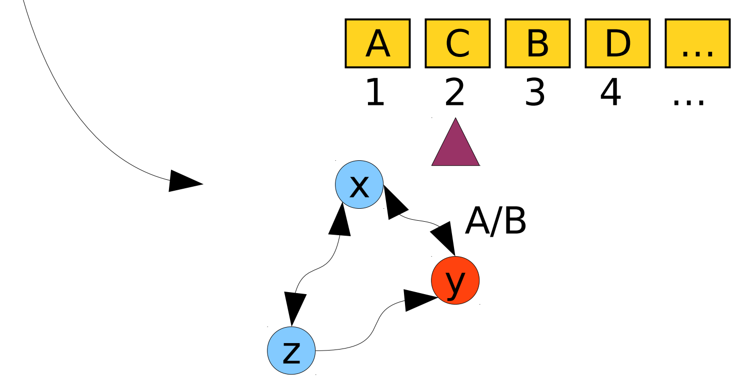
=> leakage is undecidable



Simulating a TM with an ACM



δ: (*x*, *A*) -> (*y*, *B*, L)

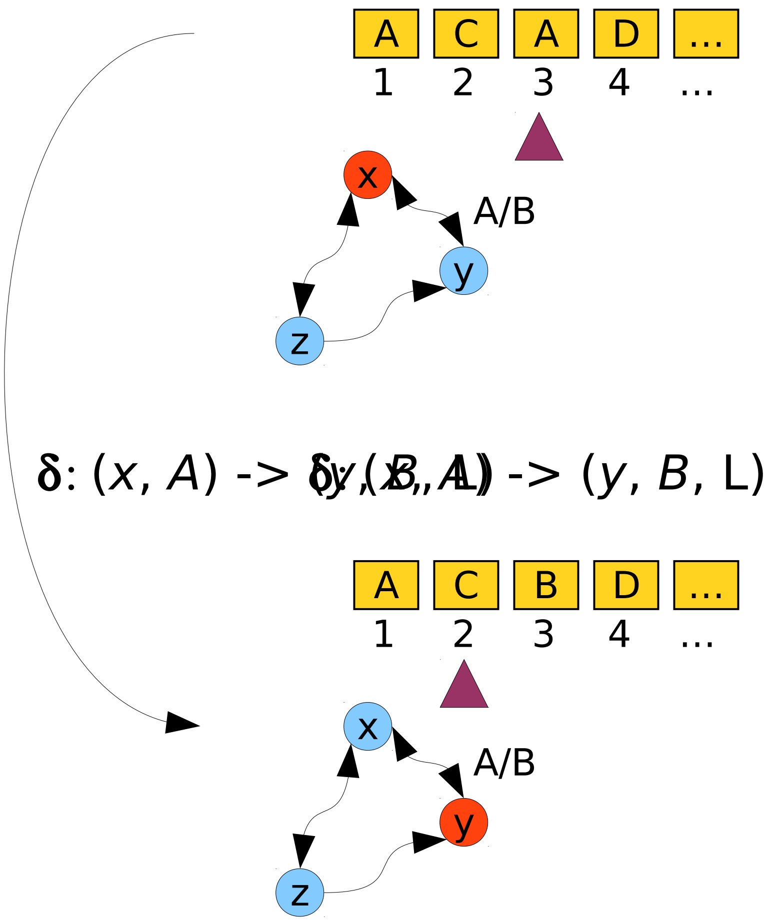


	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			A,x	
s_4			head	D

ACM Operations:

- create subject s
- create object o
- destroy subject s
- destroy object o
- enter r into (s, o)
- delete r from (s, o)

Simulating a TM with an ACM



	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			A,x	
s_4			head	D

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C,y		
s_3			B	
s_4				D

Simulating a TM with an ACM

$\boldsymbol{\delta}$: $(x, A) \text{ -> } (y, B, \text{L})$

$c_{x,A}\ (s_{head}, s_{left}) :=$

if $x \in (s_{head}, s_{head}) \wedge$

$A \in (s_{head}, s_{head})$

then

...

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			A,x	
s_4			head	D

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C,y		
s_3			B	
s_4				D

Simulating a TM with an ACM

δ: $(x, A) \rightarrow (y, B, L)$

$c_{x,A}\ (s_{head}, s_{left}) :=$

if $x \in (s_{head}, s_{head}) \wedge$

$A \in (s_{head}, s_{head})$

then

delete x,A from (s_{head}, s_{head})

...

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3				
s_4				D

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C,y		
s_3			B	
s_4				D

Simulating a TM with an ACM

δ: $(x, A) \rightarrow (y, B, L)$

$c_{x,A}$ (s_{head}, s_{left}) :=

if $x \in (s_{head}, s_{head}) \wedge$

$A \in (s_{head}, s_{head})$

then

delete x,A from (s_{head}, s_{head})

enter B into (s_{head}, s_{head})

...

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			B	
s_4				D

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C,y		
s_3			B	
s_4				D

Simulating a TM with an ACM

δ: $(x, A) \rightarrow (y, B, L)$

$c_{x,A}(s_{head}, s_{left})$:=

if $x \in (s_{head}, s_{head}) \wedge$

$A \in (s_{head}, s_{head})$

then

delete x,A from (s_{head}, s_{head})

enter B into (s_{head}, s_{head})

enter y into (s_{left}, s_{left})

endif

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C,y		
s_3			B	
s_4				D

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C,y		
s_3			B	
s_4				D

Simulating a TM with an ACM

Problem 1:

How to detect if we are at the last cell?

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			B	
s_4				D,z

Simulating a TM with an ACM

Problem 1:

How to detect if we are at the last cell?

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			B	
s_4				D,z, end

Simulating a TM with an ACM

Problem 2:

How do we know that s3 is left of s4?

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			B	
s_4				D,z, end

Simulating a TM with an ACM

Problem 2:

How do we know that s3 is left of s4?

	s_1	s_2	s_3	s_4
s_1	A	own		
s_2		C	own	
s_3			B	own
s_4				D,z, end

Exercise: write programs for all TM transitions

=> x is leaked to cell (s_i, s_i) <=> TM program halts in x with head at i

Outline

- Introduction
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model
- (Covert Channels and
  Flow Sensitive Security Type Systems)

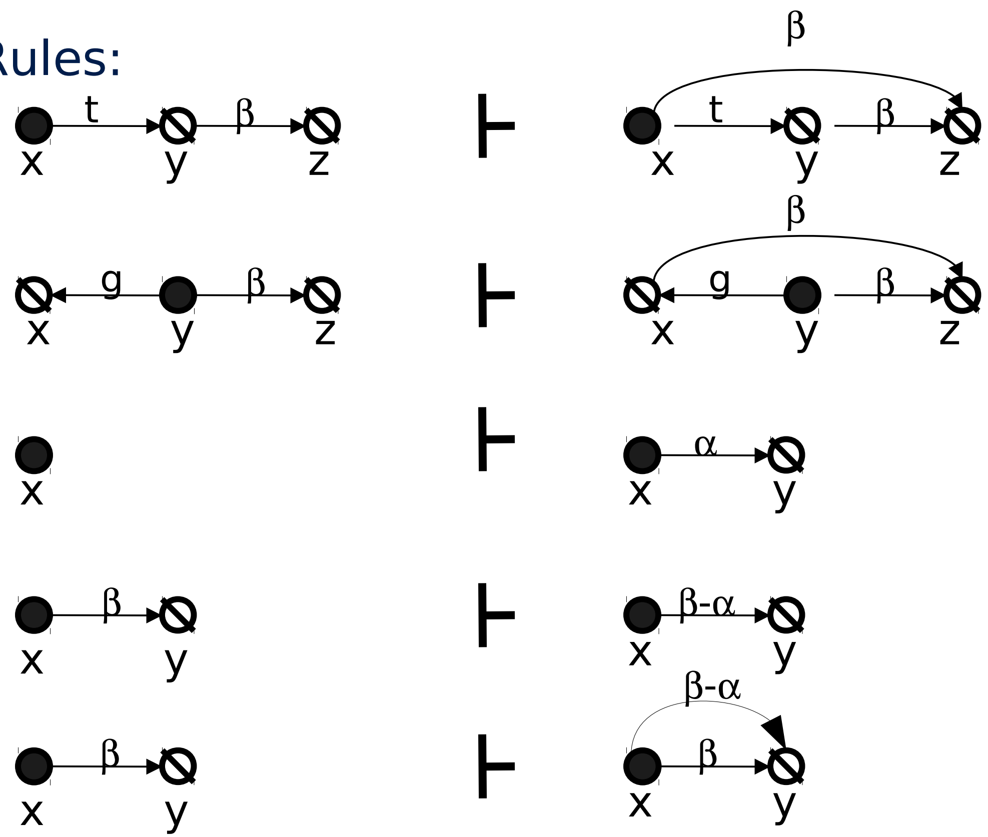
Take Grant Protection Model

Leakage in Linear Time

Vertices: object, subject (either object or subject)
Edges: r subject has capability with r right on object

Transition Rules:

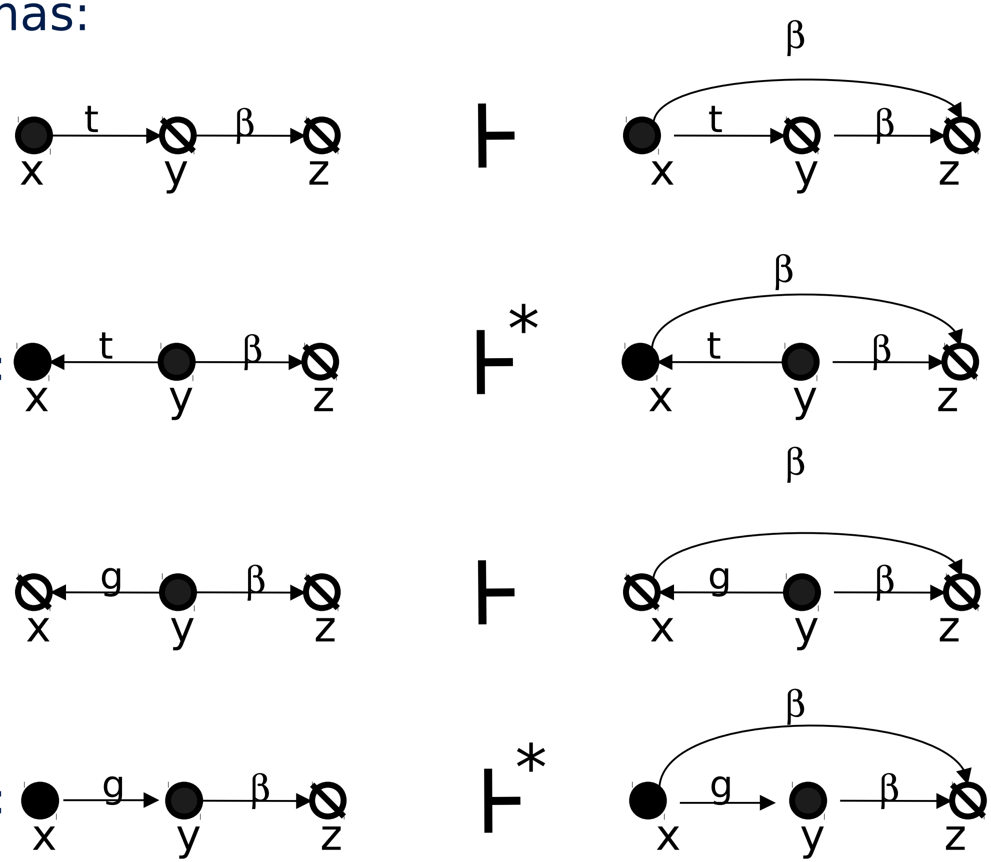
- Take
- Grant
- Create
- Remove
- Diminish



Take Grant Protection Model

A few Lemmas:

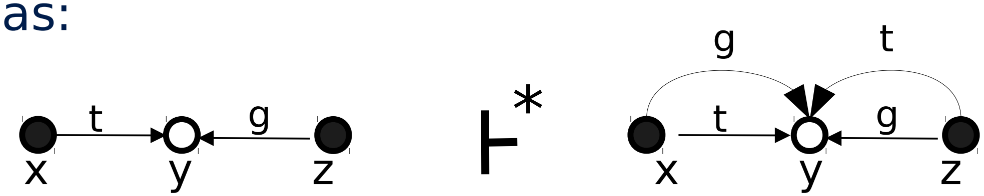
- Take
- Lemma 1:
- Grant
- Lemma 2:



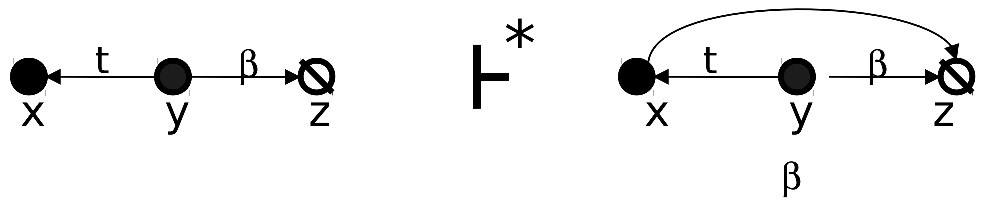
Take Grant Protection Model

A few Lemmas:

- Lemma 3:



Proof of Lemma 1



x.create	v (tg)
y.take	g on v
y.grant	β on z to v
x.take	β on z from v

Lemmas 2 and 3 are left for the exercises

Take Grant Protection Model

Theorem:

Leakage in the Take-Grant Protection Model is decidable in linear time

Proof Sketch: (decidable but not yet linear)

construct potential access graph G
apply take + grant + 3 lemmas until G does not change anymore

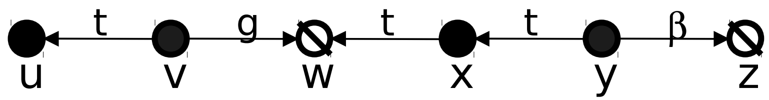
r is leaked to (s,o) if s holds (o, r) in the potential G

Note:

- delete / diminish / remove only reduce access
  => they can be omitted for the construction of G
- create introduces new entities which cannot get more privileged than their creators

Take Grant Protection Model

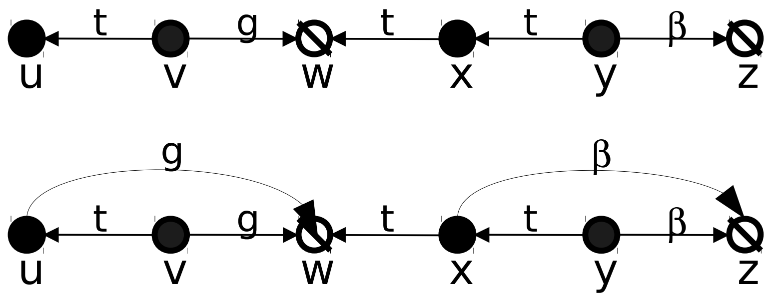
Example:



$\vdash^*$ by Lemma 1

Take Grant Protection Model

Example:

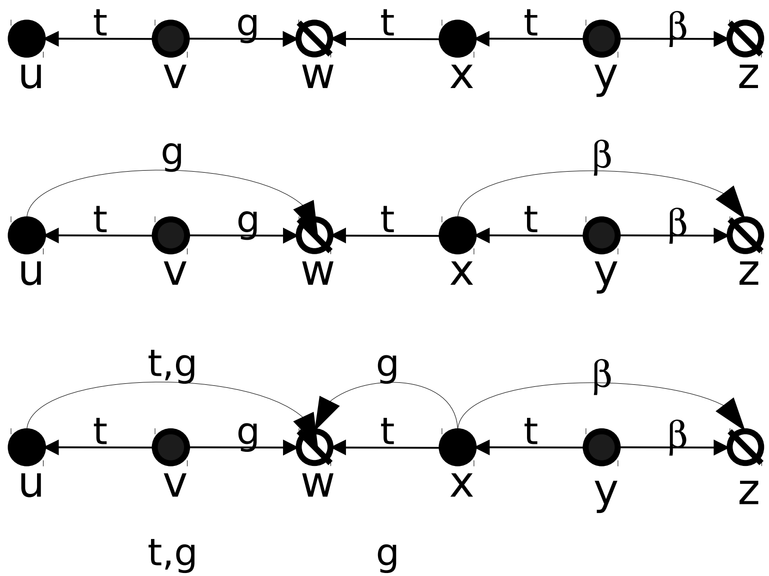


$\vdash^*$ by Lemma 1

$\vdash^*$ by Lemma 3

Take Grant Protection Model

Example:

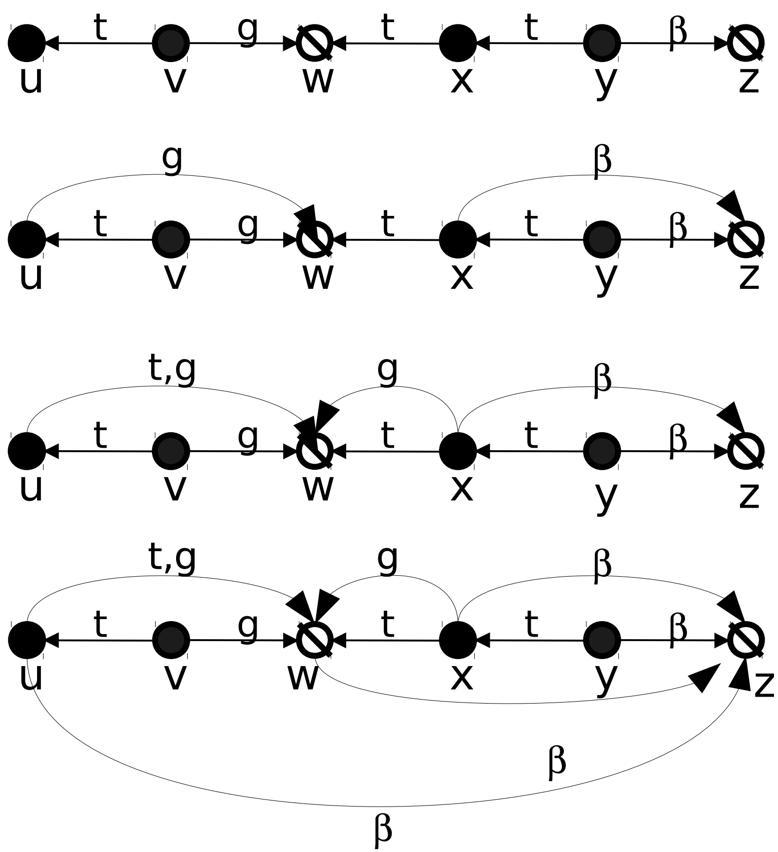


$\vdash^*$ by Lemma 1

$\vdash^*$ by Lemma 3

$\vdash^*$ x.grant β on z to h
u.take β on t from h

Example:



$\vdash^*$ by Lemma 1

$\vdash^*$ by Lemma 3

$\vdash^*$ x.grant β on z to h
u.take β on t from h

Outline

- Introduction
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model
- Covert Channels and
  Flow Sensitive Security Type Systems?
  - no slides but a little story telling

References

- B. Lampson: A note on the confinement problem
- Matt Bishop – Text Book: Computer Security – Art and Science
- P. Gallagher: A Guide to Understanding the Covert Channel Analysis of Trusted Systems [TCSEC – CC Guide]
- Proctor, Neumann: Architectural Implications of Covert Channels
- Sabelfeld, Myers: Language-based information-flow security
- Karger, Wray: Storage Channels in Disk Arm Optimizations
- Alpern, Schneider 87: Recognizing safety and lifeness
- Alves, Schneider: Enforceable security policies
- Walker, Bauer, Ligatti: More enforcable security policies
- Osvik, Shamir, Tromer: Cache Attacks and Countermeasures: the Case of AES
- Denning 67: A Lattice Model of Secure Information Flow
- Denning: Certification of programs for secure information flow.
- Hunt, Sands: On flow-sensitive security types
- Volpano, Irvine, Smith: A sound type system for secure inform. flow analysis
- Warnier: Statically checking confidentiality via dynamic labels
- Zheng, Myers: End-to-End Availability Policies and Noninterference
- Shapiro, Smith, Farber: EROS: A Fast Capability System
- Klein, Heiser + seL4: Verifying an Operating System Kernel